NO. 31 $2.00 DECEMBER 1980

MICRO™

THE 6502 JOURNAL

Multiplying on the 6502

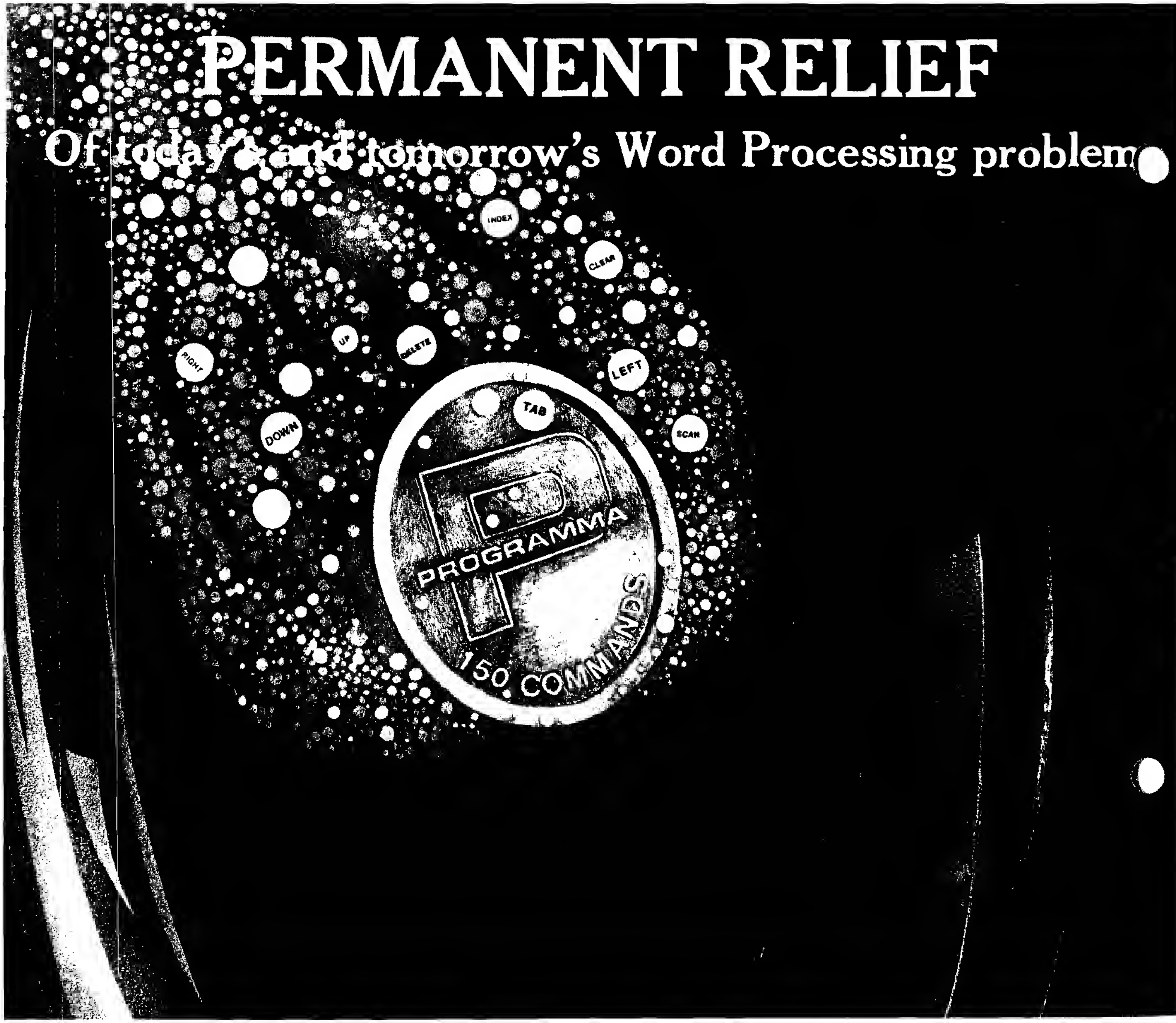
Atari Bits

Creating an Applesoft BASIC Subroutine Library

Cassette I/O for SYM BASIC

A C1P User's Notebook

STUFFIT: A Time-Saving Utility Program for PET BASIC Files



Apple PIE + Formatter

Apple PIE (Programma International Editor) and FORMAT (text formatter) offer full strength solutions to today's word processing problems. These versatile, powerful programs provide document preparation and word processing capabilities previously found only on much larger computer systems.

PIE is a general purpose, full screen editor that uses control keys and function buttons to provide a full range of editing capabilities such as search and replace, delete, copy, insert, move. Changes may be made directly anywhere on the screen and are shown as they are performed.

FORMAT uses simple instructions embedded in the input text to describe the desired appearance of the final document. It handles centering, underlining, indenting, page numbering, margins, headers, footers, even form letters, and includes a proofing capability.

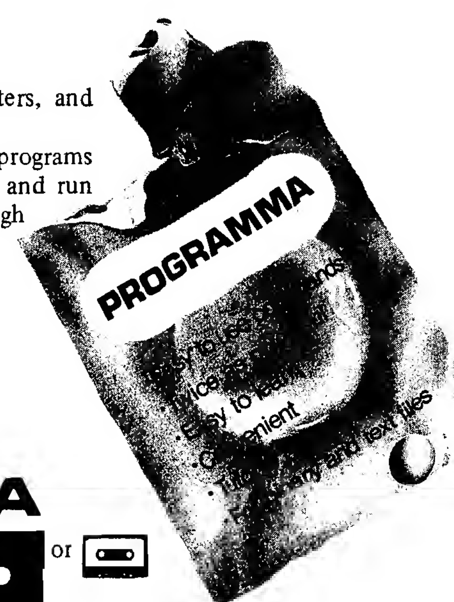
These high-quality, cost-effective programs come with comprehensive documentation and run on a 32K Apple II. They are available through your local computer store or direct from Programma International, Inc. at the introductory price of $79.95*.

VIDEX VERSION T.M.
DOUBLE VISION T.M.
SUPR TERM VERSION T.M.
STANDARD VERSION

*December 1, $129.95.

PROGRAMMA
3400 Wilshire Boulevard
Los Angeles, California 90010

or



Simple enough for the beginner. Versatile enough for the professional.

NO.31 DECEMBER 1980

MICRO™

THE 6502 JOURNAL

CONTENTS

DEPARTMENTS

MICRO is published monthly by:
MICRO INK, Inc., Chelmsford, MA 01824
Second Class postage paid at:
Chelmsford, MA 01824
Publication Number: COTR 395770
ISSN: 0271-9002

Subscription rates: U.S.	$15.00 per year
Foreign surface mail	$18.00 per year
Central America air	$27.00 per year
So. Amer/Europe air	$33.00 per year
Other air mail	$39.00 per year

For back issues, subscriptions, change of address or other information, write to:
MICRO
P.O. Box 6502
Chelmsford, MA 01824

MICRO

Editorial

Form and Substance

An Important Announcement: MICRO has not been bought by McGraw-Hill, Hayden, or anyone else. The new look is simply part of MICRO's evolution. Nor have the philosophy and purpose of MICRO changed.

Form

Since we intend to make MICRO the best journal possible, we engaged a design consultant. This issue incorporates many of his recommended changes. These include:

1. *A new MICRO logo.* Our new logo, presenting a more modern image, is intended to reflect the fact that we are one of the leaders in the microcomputer explosion.

2. *An improved cover.* Because over half our sales are through dealers, we want the cover to make a dramatic impact on the magazine rack. The "inside-the-computer-looking-out" concept has been kept, but we have improved the artwork. To help computer store customers in their browsing, we have put the issue number and date on top where they are more visible, and we have added to the bottom of the cover the titles of major articles.

3. *A new typeface.* We have changed from Helios, a very plain typeface, to a face called Trump, which we believe will be easier to read. Since the purpose of MICRO is to provide printed information, this is perhaps our most important change.

4. *Standardized listings.* In the past, we photographed and printed most listings as received. Some did not reproduce well, for example, dot-matrix printing and the AIM light blue printouts. The various sizes and shapes of listings also made for an uneven appearance. Furthermore, every assembler had different syntax features. To overcome these problems, we now have our Technical Services staff regenerate all listings. This not only results in standardized syntax and size, it also enables the staff to review the programs in detail. Listings should therefore be more accurate and easier to read.

Substance

Does this attention to form mean that we are neglecting our primary purpose—to provide information about the 6502? Not at all! The substance of MICRO is richer than ever.

1. *Improved articles.* MICRO was founded in 1977 to provide the important information which the 6502 community needed but was not getting. The backbone of MICRO has always been articles on how to use 6502-based microcomputers more effectively. Since in this field "no one knows everything and everyone knows something," we have received and published hundreds of articles from computerists of all levels of experience and from various backgrounds. Reader-generated articles will continue to be our mainstay. We are pleased with the quality and quantity of material we are receiving. Now, with an expanded technical and editorial staff, we are in a better position to work with authors, to improve their articles and to make MICRO read better than ever.

2. *Additional departments.* While articles provide important information, often not available elsewhere, their subject matter is somewhat random. Each author writes about what he feels is important. To fill the inevitable gaps, MICRO has a number of departments to cover the 6502 world systematically. Some departments have been with MICRO a long time. Others are new or still in the planning stage. The old standbys include the 6502 Bibliography, whose numbered entries have now reached the 800 mark (but have covered many thousands of individual items); The Software Catalog, which has presented many new products in its 27 appearances; and the Club Circuit, which continues to publicize 6502-based clubs. New departments have appeared recently and more are scheduled within the coming months.

3. *Enlarged staff.* Until recently, most members of the MICRO staff had to be able to handle many jobs: editing, circulation, advertising, typesetting, layout, sales, etc. While we put out a good product, it became obvious we needed more people and a better organization. To solve this problem, we have expanded and regrouped our staff. We now have separate groups responsible for Editorial, Production, Circulation, Advertising, Technical Services, and Art. With this expanded and newly organized team, the substance of MICRO will continue to be enhanced.

Robert M. Tripp

Editor/Publisher

About the Cover

Computers have played important roles in railroading for years, starting with giant mainframes, then minicomputers, and now microcomputers.

A yardmaster at Santa Fe's modern Barstow Yard may well have used a computer display like this one to direct the make up of the train. One part of the yard is called a "hump" yard, where an engine backs a string of cars over a hump for automatic sorting. As the car on the end of the string reaches the top, its number and track assignment are automatically displayed, and it is uncoupled from the other cars in front of it. The correct switches are then thrown to allow the car to roll onto the proper track. On the way down, the acceleration of the car is measured, and the proper braking force is applied to make it arrive safely. Computers are important at all stages of this operation.

The new GP-50 locomotive uses a feedback system, controlled by an on-board computer, to keep the wheels from slipping on the rails.

Microcomputers have already found their ways into model railroading, controlling yard switches, and even running several trains on a layout automatically.

Some rapid transit systems, such as BART (Bay Area Rapid Transit), in San Francisco, rely on computers for everything from collecting fares to actually running the trains.

Loren Wright, MICRO's PET Vet and resident railroad enthusiast, took this photo in March, 1979, near Bealville, California in the Tehachapi Mountains just south of Bakersfield.

GRAPHING RATIONAL FUNCTIONS

This general-purpose graphing program for a high-resolution screen—though applied here to graphing rational functions on an Apple II—is simple enough for high school students to use.

Ron Carlson
44825 Kirk Ct.
Canton, MI 48187

This is a general graphing program even though it is applied to graphing rational functions, such as

$$y = \frac{x(x-4)(x+3)}{(x-1)(x+5)}$$

If you want to graph any type of function, either remove the denominator function, FN DEN(X), or merely DEF FN DEN (X) = 1. Therefore you could graph $y = x(\sin(x))$ by the following lines:

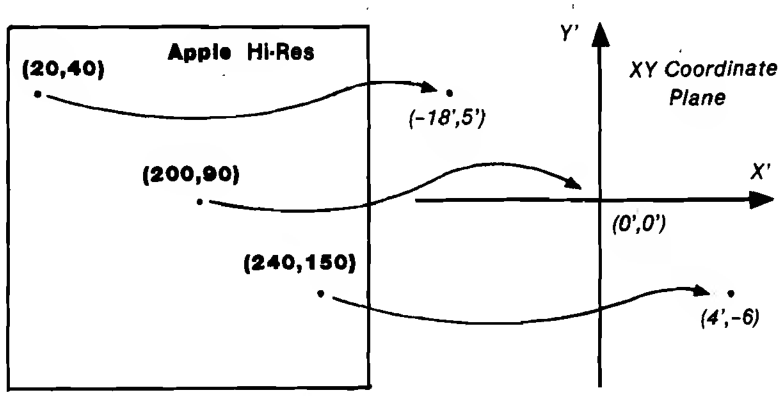
```
60 DEF FN NUM(X)=X*SIN(X)
70 DEF FN DEN(X)=1
```

This program has evolved from plotting *x*'s on a printer to the versatile graphics output of the APPLE II. Even the program on the APPLE II went through changes, ranging from graphing with an origin in the center of the screen, graphing any quadrant and choice of scale, to this version of choosing the location of the origin on the screen and the scale. High school students appear to have no difficulty using either of these options.

The program is broken into several parts: first the directions and functions section explains to the user how to define the numerator and denominator functions and how to use the program. Any legal BASIC expression can be used for the definition of the numerator and denominator. Any non-rational function can be graphed by DEF FN DEN(X) = 1. I chose the definition method of inputting the function to make the program more easily transferable to other versions of BASIC.

Another section needed for the preparation is for arrangement of the scale and determination of the location of the origin. I use the low-resolution screen with a colored cursor in the center. The user can move the cursor up, down, left, or right by using the following keys: U, D, L, R, and F when finished. The relative final position of the cursor (A,B) is changed to represent the location of the origin on the high-resolution grid of 280 × 192.

The main body of the program is the graphing section. In order to graph functions, two problems had to be overcome. The first is that the upper left corner of the screen is the origin, making it effectively upside down. The second is that I wanted to have different origins for different applications.



Sample conditions: Origin (200,90)
Scale = 10

Transformation formula: x' = (Hx − Ox)/scale
y' = (Oy − Hy)/scale
(Hx, Hy) = Apple Hi-Res coor.
(Ox, Oy) = Origin location in Hi-Res coor.

Sample points: (240,150) — (x',y')
x' = (240 − 200)/10 = 4'   (240,150) — (4', −6')
y' = (90 − 150)/10 = −6'

(20,40) — (x',y')
x' = (20 − 200)/10 = −18'   (20,40) — (−18',5')
y' = (90 − 40)/10 = 5'

(200,90) — (x',y')
x' = (200 − 200)/10 = 0'   (200,90) — (0',0')
y' = (90 − 90)/10 = 0'

Figure 1

A mathematical transformation formula will change the HGR coordinates to x and y or x and y to HGR coordinates.

(real x coor.) = (HGR x coor.) − (x coor. of origin)

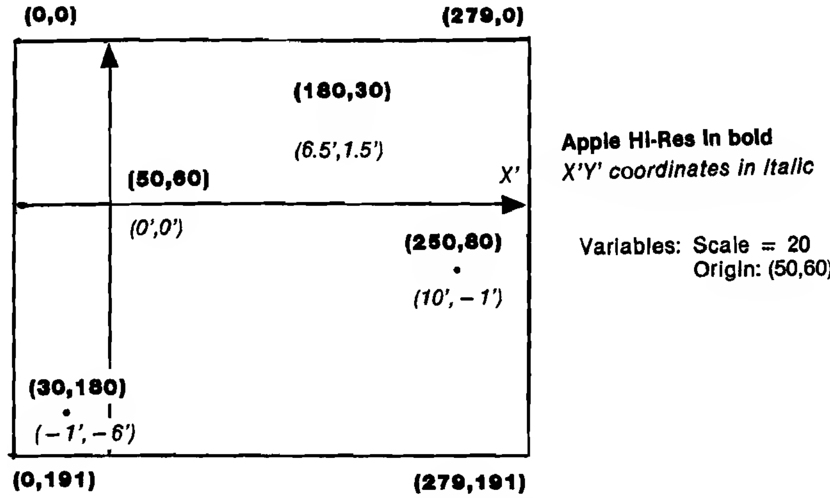
$$X = H - A$$

(real y coor.) = (y coor. of origin) − (HGR y coor.) and $Y = B - V$.

When the scale factor, S, is considered, then the transformation formulas look like:

$$X = (H-A)/S$$
$$V = B - Y*S$$

The strategy for graphing is to start H, the HGR coordinate, at 0 and continue the loop until H is 279. Translate H to the real x coordinate and substitute X into the function. Check for an asymptote, and solve for the real y-coordinate. The transformation formula will give the HGR vertical coordinate, which can be checked to make sure it is on the screen, and plot the point. When the graphing loop is finished POKE −16302,0 displays the bottom portion of the screen. The graph stays on the screen until the user depresses any key, thus giving plenty of time to make any important notes. The user is offered the choice of keeping the same function and changing the position of the origin and changing the detail by means of the scale, or starting over with a new function.



Sample points:

(30,180) — $x' = (30 - 50)/20 = -1'$, $y' = (60 - 180)/20 = -6'$ $= (-1', -6')$

(50,60) — $x' = (50 - 50)/20 = 0'$, $y' = (60 - 60)/20 = 0'$ $= (0', 0')$

(250,80) — $x' = (250 - 50)/20 = 10'$, $y' = (60 - 80)/20 = -1'$ $= (10', -1')$

(180,30) — $x' = (180 - 50)/20 = 6.5'$, $y' = (60 - 30)/20 = 1.5'$ $= (6.5', 1.5')$

Figure 2

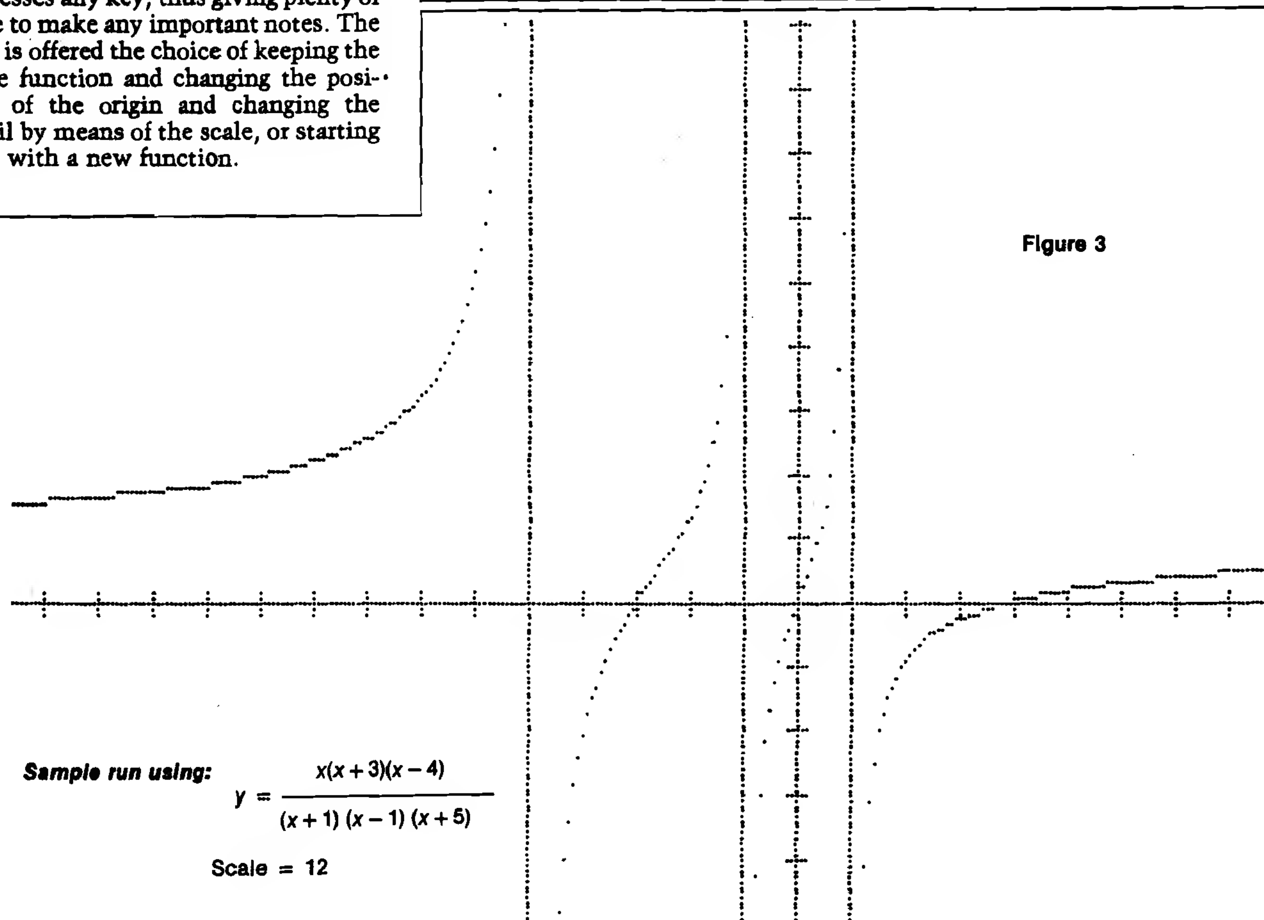


Figure 3

Sample run using:

$$y = \frac{x(x+3)(x-4)}{(x+1)(x-1)(x+5)}$$

Scale = 12

Listing 1

```
10 REM   GRAPHING RATIONAL FUNCTIONS
20 REM   BY RON CARLSON
50 REM
60 GOTO 440: REM   THE NUMERATOR FUNCTIO
N IS TO BE AT LINE 60
70 REM   PLACE DENOMINATOR FUNCTION HERE
80 REM   DEF FN DEN(X)=1 >> IF YOU HAVE
A NON-RATIONAL GRAPH
90 HOME:INPUT"THERE ARE 280 HORIZONTAL D
OTS. HOW MANY DOTS/UNIT DO YOU WANT?";S
100 VTAB 21: PRINT "INDICATE THE INTENDE
D LOCATION OF THE ORIGIN BY MOVING THE
CURSOR WITH THE   L  R  U   D  KEYS F=FI
NISHED"
110 REM  THIS ALLOWS THE USER TO SELECT
WHICH AREA OF THE GRAPH TO VIEW.
120 GOSUB 620: REM  TO POSITION THE ORIG
IN
130 REM S WILL BE THE SCALE
140 REM DETAIL INCREASES AS S INCREASES
150 VTAB 21:PRINT"AFTER THE BOTTOM HALF
OF THE GRAPH IS   FINISHED, HIT ANY KEY"
160 PRINT "THERE IS A HASH MARK ON THE A
XIS FOR    EACH UNIT"
170 HGR : HCOLOR= 7
180 REM  AXIS, WITH THE REAL ORIGIN AT (
A,B)
190 HPLOT 0,B TO 279,B: HPLOT A,0 TO A,1
91
200 REM  HASH MARKS EVERY UNIT ON THE AX
IS
210 FOR H = A TO 279 STEP S: HPLOT H,B -
 2 TO H,B + 2: NEXT
220 FOR H = A TO 0 STEP -S: HPLOT H,B -
2 TO H,B +2: NEXT
230 FOR V = B TO 191 STEP S: HPLOT A - 2
,V TO A + 2,V: NEXT
240 FOR V = B TO 0 STEP  -S: HPLOT A - 2
,V TO A + 2,V: NEXT
250 REM  ACTUAL GRAPHING
260 FOR H = 0 TO 279
270 REM  TRANSFORM THE HGR COOR TO THE R
EAL VALUE
280 X = (H - A) / S:D = FN DEN(X)
290 REM  DRAW THE VERTICAL ASYMPTOTES IF
 NECESSARY
300 IF D = 0 THEN  HCOLOR = 3: HPLOT H,0
 TO H,191: HCOLOR= 7: GOTO 350
310 Y = FN NUM(X) / D:V = B - Y * S
320 REM  TRANSFORM THE REAL Y VALUE TO H
GR AND SEE IF IT IS STILL ON THE SCREEN
330 IF V > 191 OR V < 0 THEN 350
340 HPLOT H, V
350 NEXT H
370 REM  THIS POKE WILL DISPLAY THE BOTT
OM QUARTER OF THE GRAPH
380 POKE  - 16302,0: GET A$
390 TEXT : HOME
400 INPUT :DO YOU WANT TO SHIFT THE ORIG
IN AND CHANGE SCALE? ";A$
410 IF A$ = "Y" OR A$ = "YES" THEN 90
420 GOTO 830
440 HOME : PRINT "DIRECTIONS FOR RATIONA
L FUNCTIONS"
450 PRINT"YOU MUST DEFINE YOUR FUNCTION
IN TERMS  OF NUMERATOR AND DENOMINATOR"
460 PRINT "FOR EXAMPLE IF YOU WISH TO GR
APH THE   FOLLOWING:"
470 PRINT "      (X-1)(X+2)"
480 PRINT " Y = -----------"
490 PRINT "        X(X-7)"
500 PRINT : PRINT "YOU WOULD TYPE THE FO
LLOWING"
510 PRINT "60 DEF FNNUM(X)=(X-1)*(X+2)"
520 PRINT "70 DEF FNDEN(X)=X*(X-7)"
530 PRINT "RUN"
540 PRINT : FLASH : PRINT "REMEMBER :"
550 PRINT "60 DEF FNNUM(X)=";: NORMAL :
PRINT " LEGAL BASIC EXPRESSION"
560 FLASH: PRINT "70 DEF FNDEN(X)=";: NO
RMAL: PRINT "LEGAL BASIC EXPRESSION"
570 PRINT "RUN"
580 GOTO 830
600 REM  POSITIONING THE ORIGIN ON THE S
CREEN (40 BY 40)
610 REM   USING L  R  U  D  AND  F
620 GR : COLOR= 3: PLOT 20,20:A = 20:B =
 20
630 GET A$
640 A1 = A:B1 = B
650 IF A$ = "U" THEN B = B - 1: GOTO 710
660 IF A$ = "D" THEN B = B + 1: GOTO 710
670 IF A$ = "L" THEN A = A - 1: GOTO 710
680 IF A$ = "R" THEN A = A - 1: GOTO 710
690 IF A$ = "F" THEN 800
700 REM  KEEP ON THE LO RES SCREEN
710 IF B < 1 THEN B = 1
720 IF B > 39 THEN B = 39
730 IF A < 1 THEN A = 1
740 IF A > 39 THEN A = 39
750 REM  BLANK OLD POSITION
760 COLOR= 0: PLOT A1,B1: COLOR= 3
770 REM  PLOT NEW POSITION
780 PLOT A,B
790 GOTO 630
800 A = 7 * A:B = B * 192 / 40
810 REM CHANGE SCALE TO REFLECT HGR (280
 BY 192)
820 TEXT : HOME : RETURN
830 END
```

A C1P User's Notebook

Here are "secrets" of the Challenger—one user's notes on graphics, ACIA, and tape control—which may save you time and frustration.

Robert L. Elm
446 Rothbury Ave.
Bolingbrook, IL 60439

In the last year, I have developed a notebook of things Ohio Scientific didn't tell me about my C1P. Perhaps I can share some of this information with you and save you frustrations.

Getting Started

The dealer showed me how to hook everything up, and verified that it worked. When I got home I couldn't even load the demonstration tape without errors! I was almost ready to take it back when a friend suggested I plug only one cord into the recorder at a time. Suddenly it worked! It seems my recorder feeds noise or something around within itself if the input cord is plugged in while playing a tape. The resulting interference causes errors. The solution is either to build a switch box, or simply plug in one cord at a time.

After I got by that one, I found that I could not SAVE a program without errors. This problem took a couple of days to diagnose, since I didn't know if the trouble was in the SAVE or subsequent LOAD.

I hadn't modified my T.V. monitor yet to limit the scan width so I was responding to the "TERMINAL WIDTH" prompt with "23". I noticed that the errors occurred only on long lines such as PRINT statements. Somehow the line feed at 23 characters (rather than the software supplied 24) disturbs synchronization of the LOAD routine. The solution is not to narrow the terminal width. Instead, I got busy and narrowed the T.V. scan width by making a minor adjustment in the T.V. horizontal circuitry.

Graphics

Once I started playing with the graphics, I noticed many symbols that can be added together to form larger objects. Notebook Item 1 lists those I have noted and there may be more. At any rate, it would have been nice if Ohio Scientific had told us what it had in mind with the original graphic design, rather than making all the users hunt and guess for themselves.

To see what the graphic symbols look like, you can POKE them to successive screen locations or use PRINT CHR$ (xxx). If you do the latter, you will find there are certain decimal values that will not print via CHR$. The reason is, they are control codes in ASCII for use with a printer. There are three of these:

CHR$ (7) = Bell
CHR$ (10) = Line Feed
CHR$ (13) = Carriage Return

Another handy piece of information concerns the cursor. Its value is 95 and its "home residence address" is D.54117. If you break this into the two-byte decimal format you will get 211 for the high byte and 101 for the low byte. The low byte is stored in memory at D.512 and is incremented to keep track of the cursor location as it is moved. This information should inspire some thoughts about the possibilities of cursor control in your program.

PEEKing and POKEing Around

Several authors have expanded on the memory map in the pages of MICRO so I won't go over available information. However, it would have been helpful if the operating manual had revealed where the MEMORY SIZE and TERMINAL WIDTH data are stored during a cold start. If you have something already in memory, upper memory can be reserved by POKEing a new value into D.133, D.134; and the terminal width can be changed by POKEing into D.15. This beats saving the program on tape and reloading it after another cold start.

Item 1:

Decimal Values	Graphic Object Description
5 + 6	Submarine - left side
7 + 8	Submarine - right side
9 + 10	Starship Enterprise - left side
11 + 12	Starship Enterprise - right side
181 + 182 + 179 + 6	Battleship - left side
7 + 182 + 179 + 180	Battleship - right side
232 + 234 + 235 + 232	Spacecraft
196 + 234 + 235 + 198	Spacecraft
173 + 234 + 235 + 174	Spacecraft
244 + 234 + 235 + 247	Spacecraft with aimable guns
243 + 234 + 235 + 246	Spacecraft with aimable guns
242 + 234 + 235 + 245	Spacecraft with aimable guns

Understanding The Cassette Port

Very little has been written on the internal workings of the cassette port. The memory map only states its address as hex F000-F001. This equates to decimal 61440-61441. One of my earlier programming projects was to teach my C1P to send Morse Code using the cassette port audio tone. It normally rests at 2400 cycles, and I found I could shift it to 1200 cycles and back again using POKE statements into 61440. My program ultimately worked well except that after running it, the SAVE function operated strangely and I had to warm start it to solve the problem. Investigation led back to the cassette port itself and resulted in more notebook entries.

The ACIA is a 6850 chip which is programmable and contains four accessible registers. The Transmit Data Register is a *write only* register used to send data out to tape. It is accessible using POKE 61441. The same address also contains the Receive Data Register where each word is stored as it comes in from tape. It is a *read only* register and is accessible using PEEK (61441).

The Status Register is a *read only* register you can access by PEEK (61440). Its layout is shown in Notebook Item 2. Bits 2 and 3 reflect the state of pins 23 and 24. In our circuit they are tied to ground, so they should always read zero. Bit 7 reflects the state of pin 7 which is a no-connect in the C1P. However, it can be set internally so don't write it off. The rest of the bits appear to be strictly status-reporting as defined in Item 2.

The Control Register is perhaps the most important in terms of using the cassette port in your programs. It is *writable* using POKE 61440 and it is here that I got into trouble with my Morse Code program.

The layout for this register is shown in Item 3. Bits 0 and 1 control an internal clock divider to allow a software selectable ratio of 1/1, 1/16 and 1/64. The clock supplied to the 6850 will support a 4800-baud data rate so the data rates shown in Item 3 are available.

A quick look at the C1P schematic reveals that the RS-232 circuitry is controlled by the clock output from the 6850. Therefore, if you have access to a 4800-baud printer, a quick POKE into D.61440 will allow full-speed printing! The same is true for 75 baud, if you happen to have an old "100 speed" printer lying around.

Incidentally, the Master Reset sets up a condition the SAVE software is not designed to handle. If you go to the SAVE mode while these bits are both set, the C1P will lock up waiting for the Status Register to change state. I have found that this can also happen while POKEing other values into 61440, but there is a way around it. POKE the Control Register to a decimal 3 followed immediately by the value you want. This can be done by multiple statements per line. For example, POKE 61440, 3: POKE 61440,16 will set up the 4800-baud rate.

Item 2:

ACIA Status Register — Hex F000 — Decimal 61440 — READ ONLY

7	6	5	4	3	2	1	0
IRQ	PE	OVRN	FE	$\overline{\text{CTS}}$	$\overline{\text{DCD}}$	TDRE	RDRF

RDRF - Receive Data Register Full = 1, Reset by data read or reset
TDRE - Transmit Data Register Empty = 1
DCD - Data Carrier Detect = 1 for *loss* of carrier
CTS - Clear To Send = 1 if pin 24 goes high
FE - Framing Error = 1 if first stop bit is missing indicating character synchronization error, bad transmission, or a Break condition. Bit resets when condition clears.
OVRN - OVerRuN error = 1 if RDRF = 1 when next character arrives. Reset by data read.
PE - Parity Error = 1 if parity of received data does not agree with preselected odd or even parity.
IRQ - Interrupt ReQuest = 1 to generate interrupt via pin 7. Set by $\overline{\text{DCD}}$ = 1, TDRE = 1 or RDRF = 1 depending on how TC bits of Control Register were set.

Item 3:

ACIA Control Register — Hex F000 — Decimal 61440 — WRITE ONLY

7	6	5	4	3	2	1	0
RIE	TC2	TC1	WS3	WS2	WS1	CDS2	CDS1

CDS2	CDS1	Clock Divide Select bits
0	0	1/1 (4800 Baud)
0	1	1/16 (300 Baud - normal for C1P)
1	0	1/64 (75 Baud)
1	1	Master Reset - clears Status Register

WS3	WS2	WS1	Word Select bits	
0	0	0	7 Bit + Even Parity + 2 Stop Bits	
0	0	1	7 " + Odd " + 2 " "	
0	1	0	7 " + Even " + 1 " "	
0	1	1	7 " + Odd " + 1 " "	
1	0	0	8 " + No " + 2 " "	(Normal for C1P)
1	0	1	8 " + No " + 1 " "	
1	1	0	8 " + Even " + 1 " "	
1	1	1	8 " + Odd " + 1 " "	

TC2	TC1	Transmitter Control bits
0	0	Sets RTS = 0 and inhibits transmit mode interrupt output.
0	1	Sets RTS = 0 and enables " " " "
1	0	Sets RTS = 1 and inhibits " " " "
1	1	Sets RTS = 0 and inhibits " " " " Also transmits Break (Shifts cassette port tone to 1200 cycles)

RIE Receiver Interrupt Enable = 1 to enable interrupt output in receiving mode. Allows RDRF to generate $\overline{\text{IRQ}}$.

The Word Select bits allow the variable word characteristics shown in Item 3. The C1P software sets WS3 only and supports only that configuration from tape, so I have never had a reason to change it.

The two Transmit Control bits can be used to shift the cassette port tone from 2400 to 1200 cycles and back. When both bits are set, a break is transmitted and the shift to 1200 cycles takes place. Any other value shifts it back. Normally, the C1P software initializes these bits to zeros.

The Receive Interrupt Enable bit is the last one in the Control Register. It is initialized to zero and setting it will cause a corresponding change in the Interrupt Request bit of the Status Register. I haven't found out if the C1P software uses it.

Using the above information, Item 4 shows the correct data to POKE into 61440 to get only the reaction you want.

Reading A Tape Without Loading It

Often I find I want to load more than one program into memory. Since Ohio Scientific allowed for separate LOAD and NEW commands, I only have to be sure the statement of numbers of the different programs do not overlap. A problem arises in trying to find the start of a program if there is more than one program on a tape. I can only read the tape using Load and that puts it in memory automatically, perhaps destroying what is already there.

The solution is the one line program of Item 5. I assigned it statement number 63999, so it can stay in memory and probably never get overwritten. It takes advantage of the WAIT function in Ohio Scientific 8K BASIC and capitalizes on the fact that all data is stored in ASCII form on tape. RUN 63999 gets it going for perfect copy without entering anything in memory, and "CTRL C" or RESET and a warm start stops the program when you have found the right spot on the tape. LOAD then functions normally.

I hope you are able to use some of these thoughts and comments. Perhaps you have found something that allows the completion of a stubborn program or application. If so, that is what I intended and I hope you will share your notes with the rest of us C1P users.

Item 4:

```
POKE 61440,   3 to reset Status Register (followed by POKE to valid state)
  "     "  ,  17 to initialize normal C1P state
  "     "  ,  16 for 4800 baud data rate
  "     "  ,  18 for 75 baud (100 speed) data rate
  "     "  , 113 to shift cassette port tone to 1200 cycles (actually anything
               from 96 to 127 will work as long as you shift back with a 17)
```

Item 5:

Tape Reader

```
63999 WAIT 61440,1: PRINT CHR$(PEEK (61441));: GOTO 63999
```

Drawing A Line On PET's 80 x 50 Grid

A collection of flexible, machine language routines. Allows plotting of individual points—or lines between pairs of points—using PET's quarter-box graphic characters. Additional features include erasing and screen inversion.

Harvey S. Davis
Dept. of Mathematics
Michigan State University
East Lansing, MI 48824

The purpose of this article is to provide some machine language subroutines, which are callable from BASIC, that will enable the user of a PET personal computer to draw either a point or a line on the screen, at double the resolution of the ordinary screen grid. The point or line drawn may be either plotted or erased.

An interesting feature of these subroutines is that the graphics are drawn on the screen between successive hardware updates of the screen. This means that the "snow" that results when the PET competes with the character generator video RAM will not be present. (For related programs see the interesting articles by Sherburne and Velders given in the bibliography.)

The reader should note that these subroutines are designed for use on a PET with the original ROMs. Extensive use is made of the BASIC input buffer for zero page storage. These variables would have to be relocated. Also there are calls to $E840 to determine the status of the update of the screen.

Machine Language Subroutines

```
             ADLO     *     $0001
             ADHI     *     $0002
             XCHR     *     $0033 WRITTEN FOR 2.0 ROM'S
             XADL     *     $0034 MOST OF THESE PAGE
             XADH     *     $0035 ZERO ADDRESSES MUST
             BFRNXT   *     $0036 BE CHANGED FOR 3.0
             BFFRLO   *     $0037 BASIC ROM'S.
             BFFRHI   *     $0038
             COUNT    *     $0039
             XO       *     $003A
             YO       *     $003B
             XI       *     $003C
             YI       *     $003D
             DX       *     $003E
             DY       *     $003F
             XLO      *     $0040
             XHI      *     $0041
             YLO      *     $0042
             YHI      *     $0043
             XVAL     *     $0044
             YVAL     *     $0045
             ZALO     *     $0046
             ZAHI     *     $0047
             QUAD     *     $0048
             FLAG     *     $0049
             ZTABLE   *     $004A
             VIAADD   *     $E840

1B00 20 88 1F   SNGLPT   JSR    FRSTPT CALLED BY BASIC--
1B03 20 F9 1B            JSR    BFROUT DRAWS SINGLE POINT AT
1B06 60                  RTS           (XVAL),(YVAL).

1B10 20 00 1F   LINE     JSR    AIM    CALLED BY BASIC--
1B13 20 38 1F            JSR    PREDRW DRAWS LINE FROM
1B16 20 88 1F            JSR    FRSTPT XO,YO TO X1,Y1.
1B19 20 60 1F            JSR    DRAW
1B1C 20 F9 1B            JSR    BFROUT
1B1F 60                  RTS

1BF9 AD 40 E8   BFROUT   LDA    VIAADD WAITS FOR
1BFC 29 20               AND    #$20   HARDWARE UPDATE
1BFE D0 F9               BNE    BFROUT OF SCREEN
1C00 60         BUFFER   RTS           SUBROUTINE CREATED
                                       BY "BFFRIN"
                                       STARTS AT $1C00.
```

Display Grids

The PET displays characters on its screen with reference to a 25 x 40 grid. Each screen location is associated with an address in memory, according to the following formula:

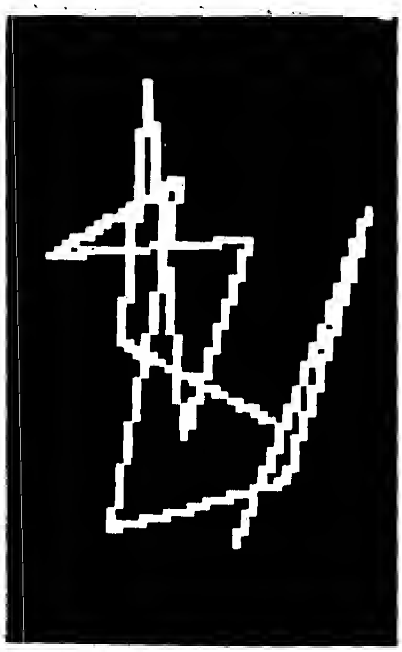
$$ADDRESS = 32*1024 + 25*Y + X$$

X denotes the column ($0 \le X \le 39$) and Y denotes the row ($0 \le Y \le 24$). The positive x-axis points to the left, while the positive y-axis points down. Thus address 32*1024 is associated with the upper left-hand corner of the screen. I will refer to these conventions as the "screen grid".

In order to discuss the generation of graphics having twice the resolution of the screen grid, I shall use two other grids which I shall call the observer's grids.

The observer's grid positive down is an 80 x 50 grid ($0 \le X \le 79$, $0 \le Y \le 49$) with the origin in the upper left hand corner, the x-axis proceeding from left to right and the y-axis proceeding from top to bottom.

The observer's grid positive up is an 80 x 50 grid ($0 \le X \le 79$, $0 \le Y \le 49$) with the origin in the lower left hand corner, the x-axis proceeding from left to right and the y-axis proceeding from bottom to top.



(continued)

```
1E00 20          TABLE    =    $20        THIS IS A TABLE OF
1E01 6C                   =    $6C        ALL SIXTEEN POSSIBLE
1E02 7B                   =    $7B        QUARTER-BOX CHAR.'S.
1E03 62                   =    $62
1E04 7C                   =    $7C
1E05 E1                   =    $E1
1E06 FF                   =    $FF
1E07 FE                   =    $FE
1E08 7E                   =    $7E
1E09 7F                   =    $7F
1E0A 61                   =    $61
1E0B FC                   =    $FC
1E0C E2                   =    $E2
1E0D FB                   =    $FB
1E0E EC                   =    $EC
1E0F A0                   =    $A0
1E10 A2 0F       INIT     LDX  #$0F       MOVES ABOVE TABLE
1E12 BD 00 1E    LOAD     LDA  TABLE,X    TO PAGE ZERO
1E15 95 4A                STA  ZTABLE,X   BEGINNING AT
1E17 CA                   DEX             ZTABLE=$004A
1E18 10 F8                BPL  LOAD
1E1A 60          RETURN   RTS
1E1B A5 44       POINT    LDA  XVAL       COMPUTES SCREEN
1E1D 30 FB                BMI  RETURN     ADDRESS AND ENCODES
1E1F C9 50                CMP  #$50       CHARACTER QUADRANT
1E21 10 F7                BPL  RETURN
1E23 A5 45                LDA  YVAL
1E25 30 F3                BMI  RETURN     TESTS FOR XVAL AND
1E27 C9 32                CMP  #$32       YVAL IN RANGE
1E29 10 EF                BPL  RETURN
1E2B 24 49       ORIENT   BIT  FLAG       CHECKS BIT 7 OF FLAG
1E2D 10 07                BPL  PREP       = 0 NO CHANGE
1E2F A9 31                LDA  #$31       =1 YVAL = 49 - YVAL
1E31 38                   SEC
1E32 E5 45                SBC  YVAL
1E34 85 45                STA  YVAL
1E36 18          PREP     CLC             2 * PARITY(YVAL) +
1E37 A5 45                LDA  YVAL       PARITY(XVAL)
1E39 29 01                AND  #$01       TRANSFERRED TO
1E3B 0A                   ASL  A          X REG.
1E3C 85 46                STA  ZALO
1E3E A5 44                LDA  XVAL
1E40 29 01                AND  #$01
1E42 65 46                ADC  ZALO
1E44 AA                   TAX
1E45 A9 10                LDA  #$10       QUAD = 2   (4-XR+1)
1E47 4A          LOOPA    LSR  A
1E48 CA                   DEX
1E49 10 FC                BPL  LOOPA
1E4B 85 48                STA  QUAD
1E4D 46 44                LSR  XVAL       XVAL = INT(XVAL/2)
1E4F 46 45                LSR  YVAL       YVAL = INT(YVAL/2)
1E51 A9 80       LOCAD    LDA  #$80       SET ADLO,ADHI FOR
1E53 85 02                STA  ADHI       SCREEN UPPER LEFT
1E55 A9 00                LDA  #$00       ZALO = 8 * YVAL
1E57 85 01                STA  ADLO
1E59 85 47                STA  ZAHI
1E5B A5 45                LDA  YVAL
1E5D 0A                   ASL  A
```

```
1E5E 0A               ASL   A
1E5F 0A               ASL   A
1E60 85 46            STA   ZALO      ZAHI,ZALO =
1E62 06 46            ASL   ZALO        4 * ZALO
1E64 26 47            ROL   ZAHI        32 * YVAL
1E66 06 46            ASL   ZALO
1E68 26 47            ROL   ZAHI
1E6A 65 44            ADC   XVAL      ZAHI,ZALO =
1E6C 65 46            ADC   ZALO        ACCUM. + XVAL +
1E6E 85 46            STA   ZALO        ZAHI,ZALO
1E70 90 02            BCC   ADDAD       = 40 * YVAL + XVAL
1E72 E6 47            INC   ZAHI
1E74 18      ADDAD    CLC              + ADHI,ADLO
1E75 A5 46            LDA   ZALO       = $8000 + 40 * YVAL
1E77 65 01            ADC   ADLO       + XVAL
1E79 85 01            STA   ADLO
1E7B A5 47            LDA   ZAHI
1E7D 65 02            ADC   ADHI
1E7F 85 02            STA   ADHI
1E81 60               RTS

1E88 AD 40 E8 GETCHR  LDA   VIAADD    WAITS FOR HARDWARE
1E8B 29 20            AND   #$20      SCREEN UPDATE--
1E8D D0 F9            BNE   GETCHR    AVOIDS SNOW!!
1E8F A2 00            LDX   #$00      GET CHARACTER FROM
1E91 A1 01            LDA   (ADLO,X)  SCREEN ADLO,ADHI
1E93 85 33            STA   XCHR      STASH IN XCHR
1E95 A5 01            LDA   ADLO      XADL = ADLO
1E97 85 34            STA   XADL
1E99 A5 02            LDA   ADHI      XADH = ADHI
1E9B 85 35            STA   XADH
1E9D 60               RTS

1EA0 A5 33   UPDATE   LDA   XCHR      LOOK FOR XCHR IN
1EA2 A2 00            LDX   #$00      ZTABLE
1EA4 D5 4A   LOOPB    CMP   ZTABLE,X
1EA6 F0 06            BEQ   TSTFLG    FOUND
1EA8 E8               INX
1EA9 E0 10            CPX   #$10      AT END OF TABLE?
1EAB D0 F7            BNE   LOOPB
1EAD 60               RTS
1EAE 24 49   TSTFLG   BIT   FLAG      TEST FLAG--BIT 6
1EB0 50 0C            BVC   PLOT       =0 PLOT =1 ERASE
1EB2 A9 FF   ERASE    LDA   #$FF      BLANKS QUANDRANT
1EB4 45 48            EOR   QUAD      INDICATED BY
1EB6 85 48            STA   QUAD      (QUAD) OF COVERING
1EB8 8A               TXA             CHARACTER (SCHR).
1EB9 25 48            AND   QUAD      TABLE INDEX OF NEW
1EBB AA               TAX             CHAR. NOW IN XR.
1EBC 10 04            BPL   CONT      ACTUALLY UNCOND.!
1EBE 8A      PLOT     TXA             FILLS QUADRANT
1EBF 05 48            ORA   QUAD      INDICATED BY QUAD.
1EC1 AA               TAX
1EC2 B5 4A   CONT     LDA   ZTABLE,X  STORES NEW
1EC4 85 33            STA   XCHR      COVERING CHAR.
1EC6 60               RTS
```

(continued)

Character Graphics

The character that is displayed at a given location on the screen is completely determined by the byte value stored at the corresponding address.

Among the characters that the PET makes available are 16 that allow the user effectively to double the resolution simultaneously in each direction. These characters can be visualized as a square with each of its quadrants either blank or lit. Table 1 gives these characters and their byte equivalents.

Table 1

a	b	c	d	value
0	0	0	0	20
0	0	0	1	6C
0	0	1	0	7B
0	0	1	1	62
0	1	0	0	7C
0	1	0	1	E1
0	1	1	0	FF
0	1	1	1	FF

a	b	c	d	value
1	0	0	0	7E
1	0	0	1	7F
1	0	1	0	61
1	0	1	1	Fc
1	1	0	0	E2
1	1	0	1	FB
1	1	1	0	EC
1	1	1	1	AO

a	b
c	d

a = 0 if and only if quadrant a is blank
b = 0 if and only if quadrant b is blank
c = 0 if and only if quadrant c is blank
d = 0 if and only if quadrant d is blank

Some Notation Conventions

If XXX denotes an address, then [XXX] will denote the byte value stored at that address. [XXX] may also, in the appropriate context, denote the character represented by that byte value.

If XXX and YYY denote addresses, then [XXX,YYY] will denote the address whose low byte is [XXX] and whose high byte is [YYY]. [XXX,YYY] will denote the double precision number whose most significant byte is [XXX] and whose least significant byte is [YYY].

AC, XR, and YR denote the byte values at the accumulator, the *x* register and the *y* register.

BASIC Subroutines (for drawing a single point at X,Y on the observer's grid)

1. Set the observer's grid positive down. This is done by setting the most significant bit of FLAG = $0049 = 73 to 0.

```
10 POKE 73, PEEK (73) AND 127
: RETURN
```

2. Set the observer's grid positive up. This is done by setting the most significant bit of FLAG = $0049 = 73 to 1.

```
15 POKE 73, PEEK (73) OR 128 :
RETURN
```

3. Set draw mode to PLOT. This is done by setting the second most significant bit of FLAG to 0.

```
20 POKE 73, PEEK (73) AND 191
: RETURN
```

4. Set draw mode to ERASE. This is done by setting the second most significant bit of FLAG to 1.

```
25 POKE 73, PEEK (73) OR 64 :
RETURN
```

5. Draw the point at X,Y. This is done by setting [XVAL] = X, [YVAL] = Y and calling subroutine SNGLPT.

```
30 POKE 68,X : POKE 69,Y : SYS
(6912) : RETURN
```

(continued)

```
1EC8 A9 00     SETBFR  LDA   #$00        ADJUSTS
1ECA 85 36             STA   BFRNXT      BUFFER
1ECC 85 37             STA   BFFRLO      POINTERS
1ECE A9 1C             LDA   #$1C
1ED0 85 38             STA   BFFRHI
1ED2 60                RTS

1ED8 A4 36     BFRIN   LDY   BFRNXT      BUFFER INDEX
1EDA A9 A9             LDA   #$A9         = "LDA #"
1EDC 91 37             STA   (BFFRLO),Y
1EDE C8                INY
1EDF A5 33             LDA   XCHR        XCHAR IN NEXT
1EE1 91 37             STA   (BFFRLO),Y  BUFFER LOC.
1EE3 C8                INY
1EE4 A9 8D             LDA   #$8D         = "STA ABS."
1EE6 91 37             STA   (BFFRLO),Y
1EE8 C8                INY
1EE9 A5 34             LDA   XADL        TRANSFER CALC.
1EEB 91 37             STA   (BFFRLO),Y  SCREEN ADDRESS
1EED C8                INY               TO BUFFER PROG.
1EEE A5 35             LDA   XADH
1EF0 91 37             STA   (BFFRLO),Y
1EF2 C8                INY
1EF3 A9 60             LDA   #$60         = "RTS"
1EF5 91 37             STA   (BFFRLO),Y  BFRNXT POINTS AT
1EF7 84 36             STY   BFRNXT      THIS "RTS".
1EF9 60                RTS

1F00 A9 80     AIM     LDA   #$80    SETS POSITIVE
1F02 05 49             ORA   FLAG    DOWN
1F04 85 49             STA   FLAG
1F06 A5 3C     CMPXI   LDA   XI      MAKES SURE PLOT
1F08 C5 3A             CMP   XO      PROCEEDS IN
1F0A 10 0E             BPL   CMPYI   POSITIVE X
1F0C A6 3A             LDX   XO      DIRECTION
1F0E 85 3A             STA   XO
1F10 86 3C             STX   XI
1F12 A5 3D             LDA   YI
1F14 A6 3B             LDX   YO
1F16 85 3B             STA   YO
1F18 86 3D             STX   YI
1F1A A5 3D     CMPYI   LDA   YI      MAKES SURE PLOT
1F1C C5 3B             CMP   YO      CALCULATIONS
1F1E 10 12             BPL   RTNAIM  PROCEED IN POSI-
1F20 38                SEC           TIVE Y DIRECTION,
1F21 A9 31             LDA   #$31    IN SPITE OF ACTUAL
1F23 E5 3D             SBC   YI      ORIENTATION.
1F25 85 3D             STA   YI
1F27 38                SEC
1F28 A9 31             LDA   #$31
1F2A E5 3B             SBC   YO
1F2C 85 3B             STA   YO
1F2E 06 49             ASL   FLAG
1F30 46 49             LSR   FLAG
1F32 60        RTNAIM  RTS

1F38 38        PREDRW  SEC           INITIALIZES
1F39 A5 3C             LDA   XI      VARIABLES USED
1F3B E5 3A             SBC   XO      IN LINE
1F3D 85 3E             STA   DX      DRAWING
1F3F 38                SEC
```

```
1F40 A5 3D          LDA   Y1
1F42 E5 3B          SBC   Y0
1F44 85 3F          STA   DY
1F46 A5 3A          LDA   X0
1F48 85 44          STA   XVAL
1F4A 85 41          STA   XHI
1F4C A5 3B          LDA   Y0
1F4E 85 45          STA   YVAL
1F50 85 43          STA   YHI
1F52 A2 80          LDX   #$80   PLOTS LINE
1F54 86 40          STX   XLO    FROM X0,Y0 TO
1F56 86 42          STX   YLO    X1,Y1 BY
1F58 A2 00          LDX   #$00   COMPUTING AND
1F5A 86 39          STX   COUNT  PLOTTING
1F5C 60             RTS          256 CONSECUTIVE
                                 POINTS
1F60 A5 3E    DRAW   LDA   DX
1F62 18              CLC
1F63 65 40           ADC   XLO
1F65 85 40           STA   XLO
1F67 90 02           BCC   INCRY
1F69 E6 41           INC   XHI
1F6B A5 3F    INCRY  LDA   DY     PLOTS POINT AT
1F6D 18              CLC          XVAL,YVAL.
1F6E 65 42           ADC   YLO
1F70 85 42           STA   YLO
1F72 90 02           BCC   CALLPT INITIALIZE AND
1F74 E6 43           INC   YHI    PLOT FIRST
1F76 A5 41    CALLPT LDA   XHI    POINT.
1F78 85 44           STA   XVAL   IS THIS ADDRESS
1F7A A5 43           LDA   YHI    SAME AS THAT
1F7C 85 45           STA   YVAL   OF LAST POINT?
1F7E 20 A0 1F        JSR   NEXTPT
1F81 E6 39           INC   COUNT
1F83 D0 DB           BNE   DRAW   END OF BUFFER?
1F85 4C D8 1E        JMP   BFRIN
1F88 20 10 1E FRSTPT JSR   INIT
1F8B 20 1B 1E        JSR   POINT
1F8E 20 88 1E        JSR   GETCHR BUFFER TO SCREEN
1F91 20 A0 1E        JSR   UPDATE RESET TO NOMINAL
1F94 20 C8 1E        JSR   SETBFR LAST CHR. TO BUFF.
1F97 20 D8 1E        JSR   BFRIN  CHR. AT NEXT ADDR.
1F9A 60              RTS

1FA0 20 1B 1E NEXTPT JSR   POINT
1FA3 A5 01           LDA   ADLO
1FA5 C5 34           CMP   XADL
1FA7 D0 06           BNE   NEWADR
1FA9 A5 02           LDA   ADHI
1FAB C5 35           CMP   XADH
1FAD F0 13           BEQ   TOUPDT
1FAF A9 F5    NEWADR LDA   #$F5
1FB1 38              SEC
1FB2 C5 36           CMP   BFRNXT
1FB4 B0 06           BCS   TOBFFR
1FB6 20 F9 1B        JSR   BFROUT
1FB9 20 C8 1E        JSR   SETBFR
1FBC 20 D8 1E TOBFFR JSR   BFRIN
1FBF 20 88 1E        JSR   GETCHR
1FC2 20 A0 1E TOUPDT JSR   UPDATE
1FC5 60              RTS
```

To use the BASIC subroutine for drawing a line from X,Y to U,V on the observer's grid positive up, the draw mode may be set by BASIC subroutines 20 and 25 above. The drawing is done by setting [X0] = X,[Y0] = Y,[X1] = U,[Y1] = V and calling subroutine LINE.

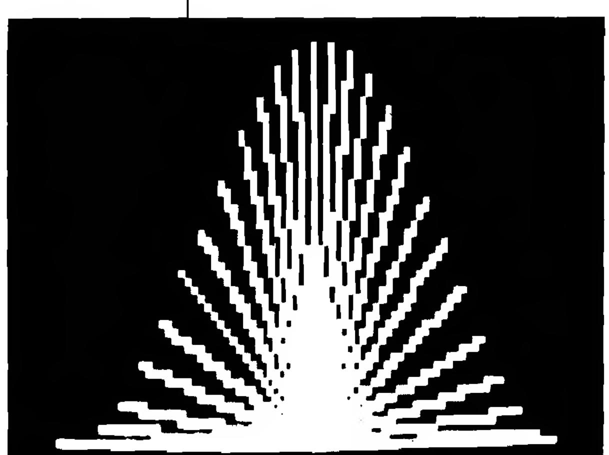
```
40 POKE 58,X : POKE 59,Y :
POKE 60,U : POKE 61,V

42 SYS(6928) : RETURN
```

The user should note that if a BASIC program intermixes point and line commands, the orientation of the observer's grid must be reset by either subroutine 10 or 15, before each point command that immediately follows a line command. It need not be reset between successive point commands. The line command [subroutine 40] always assumes that the observer's grid is positive up.

Bibliography

Sherburne, J.R., *High Resolution Plotting for the PET, MICRO* [10:19].

Velders, J.A., *Hi-resolution Plot*, Pet User Notes, vol. 2, #1, p. 18.



A Random-Character Morse Code Teacher For The AIM 65

High-speed Morse Code character recognition comes easily when you program your computer to generate code sounds at 13 words per minute and up.

Eugene V. Weiner
511 Fifth Avenue
Coralville, IA 52241
with
Marvin L. DeJong
Dept. of Math & Physics
The School of the Ozarks
Point Lookout, MO 65726
and
Russell V. Lenth
Dept. of Statistics
University of Iowa
Iowa City, IA 52242

In September 1979 I noticed an article[1] by M.L. DeJong in MICRO (16:29). There was also a Morse Code sending software[2] offered by the above author. Both were of great interest to me because one of several goals I am working toward is the use of my AIM 65 to send and receive Morse Code. I also wanted to use the AIM 65 to send code lessons.

Some years ago in learning the Morse Code, I decided on a technique for learning and teaching it. By sending the individual characters at a rate greater than 13 w.p.m. (words per minute), with increased spacing to lower the presentation rate, the student would be able to learn the actual character sound sooner, rather than taking the character apart mentally, counting the various dots and dashes, and then deciding which character was sent. At higher code transmission rates (13 w.p.m. and up) character recognition has to be virtually instantaneous.

An accompanying learning problem is lesson material. A recording or tape of text (even radom groups) is soon memorized and its effectiveness is lost. Randomly generated character groups are difficult to memorize and give no clues as to what character may be heard next. I thought this whole approach could be accomplished with a small computer. All that was needed was some software and a little hardware.

An article[3] appeared in MICRO February 1980 (21:19) by M.L. DeJong that described a Morse Code send and receive program. I was so impressed that I felt it was time to press forward and have available a random code program to provide the additional tool for teaching Morse Code. I discussed it with a local Amateur Radio operator, Russell V. Lenth, who is quite conversant in BASIC. On short notice he produced a program to generate random character groups, but not in Morse Code. Neither of us knew how to join it with a Morse Code generation program, so I sent a letter to M.L. DeJong detailing what we wished to accomplish, and included the BASIC program. He was kind enough to write another BASIC program and a machine language program which generally accomplished what was desired. We (R.V. Lenth and I) worked over the two programs to do a number of additional things. Below are listed the functions of the refined program.

1. Characters are generated at 15 w.p.m. or faster.
2. Characters are presented in groups of 5.
3. User has option to input specific characters for random sending.
4. Program provides for the 42 characters available in the look-up Table[1, 2].
5. Spacing between characters is varied below 15 w.p.m. to control presentation rate.
6. Program prints each line of characters after the entire line is sent.

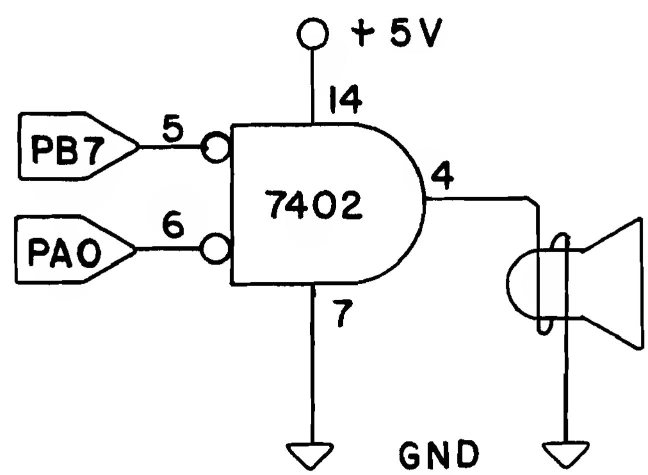


***Figure 1:* This circuit will produce an easy-to-listen-to sound.**

The hardware to generate the code sounds and/or operate an external relay for keying a transmitter or code practice oscillator was taken from several articles by M.L. DeJong[1, 2, 3]. We believe there are sufficient hardware options to cover most users' needs. An additional option is shown in figure 1 from a private note from M.L. DeJong, modified by E.V. Weiner.

The machine language program which generates the Morse Code is relatively short. When activating BASIC, limit memory to 2048 bytes. The look-up table is located above $0F20. The Morse Code send routine starts at $0F5C.

Look Up Table

```
0F20 00 00 00 00
0F24 00 00 00 CE
0F28 00 00 00 00
0F2C CE 8C 56 94
0F30 FC 7C 3C 1C
0F34 0C 04 84 C4
0F38 E4 F4 16 32
0F3C 00 8C 00 32
0F40 00 60 88 A8
0F44 90 40 28 D0
0F48 08 20 78 B0
0F4C 48 E0 A0 F0
0F50 68 D8 50 10
0F54 C0 30 18 70
0F58 98 B8 C8 00
```

```
          RANDOM CHARACTER MORSE CODE

          BY EUGENE V. WEINER,
             MARVIN L. DEJONG
             RUSSELL V. LENTH

          FOR THE AIM 65


AIM EQUATES

UDRAH    *  $A001   DATA REG A
UDDRA    *  $A003   DATA DIR REG A
UTIL     *  $A004   TIMER 1 COUNTER LOW
UTILA    *  $A005   TIMER 1 COUNTER HIGH
UACR     *  $A00B   AUX CONTROL REGISTER
DIV      *  $A497   DIVIDE BY 1024 TIMER

AIM SUBROUTINES

OUTPUT   *  $E97A   OUTPUT A CHARACTER
XXXX     *  $C0D1

PROGRAM EQUATE

TABLE    *  $0F00

                        ORG   $0F5C

0F5C A9 81     START    LDA   #$81
0F5E 8D 01 A0           STA   UDRAH
0F61 8D 03 A0           STA   UDDRA
0F64 A9 C0              LDA   #$C0
0F66 8D 0B A0           STA   UACR
0F69 A9 00              LDA   #$00
0F6B 8D 04 A0           STA   UTIL
0F6E A9 04              LDA   #$04
0F70 8D 05 A0           STA   UTILA
0F73 A5 38              LDA   $0038
0F75 48                 PHA
0F76 A8                 TAY
0F77 AD 00 0F           LDA   TABLE
0F7A F0 21              BEQ   LBLF
0F7C 0A        LBLA     ASL   A
0F7D F0 0F              BEQ   LBLD
0F7F 48                 PHA
0F80 B0 06              BCS   LBLB
0F82 20 A2 0F           JSR   LBLG
0F85 4C 8B 0F           JMP   LBLC
0F88 20 BB 0F  LBLB     JSR   LBLK
0F8B 68        LBLC     PLA
0F8C D0 EE              BNE   LBLA
0F8E A0 02     LBLD     LDY   #$02
0F90 20 C0 0F  LBLE     JSR   LBLL
0F93 88                 DEY
0F94 D0 FA              BNE   LBLE
0F96 68                 PLA
0F97 20 7A E9           JSR   OUTPUT
0F9A 4C D1 C0           JMP   XXXX
0F9D A0 04     LBLF     LDY   #$04
0F9F 4C 90 0F           JMP   LBLE
0FA2 A0 01     LBLG     LDY   #$01
0FA4 CE 01 A0  LBLH     DEC   UDRAH
0FA7 20 C0 0F  LBLI     JSR   LBLL
0FAA 88                 DEY
0FAB D0 FA              BNE   LBLI
0FAD AD 01 A0           LDA   UDRAH
0FB0 4A                 LSR   A
0FB1 B0 07              BCS   LBLJ
0FB3 EE 01 A0           INC   UDRAH
0FB6 C8                 INY
0FB7 4C A7 0F           JMP   LBLI
0FBA 60        LBLJ     RTS
0FBB A0 03     LBLK     LDY   #$03
0FBD 4C A4 0F           JMP   LBLH
0FC0 A5 37     LBLL     LDA   $0037
0FC2 8D 97 A4           STA   DIV
0FC5 2C 97 A4  LBLM     BIT   DIV
0FC8 10 FB              BPL   LBLM
0FCA 60                 RTS
```

The basic program allocates memory for the characters to be sent in statements 10 and 20.

- 30 to 50 generate the numbers.
- 60 to 80 generate the letters.
- 100 defaults to all the characters in the look-up table.
- 110 to 140 input desired characters other than default set.
- Speed is entered at 150.
- 151 to 180 provide for character spacing.
- 190 to 200 enter speed values into machine language subroutine.
- 210 to 220 specify starting address for the machine language subroutine.
- 230 to 260 fill the "A" array with 20 random characters.
- 270 to 400 send and display the characters.
- 290 puts characters into memory.
- 300 to 310 generate character spacing.
- 320 sends and displays the characters.
- 330 prints 20 characters after they are sent and displayed.
- 360 to 390 put timed spaces between groups of 5 characters.
- Starting at 410 is a subroutine which determines how many lines of 20 characters are to be sent before stopping and re-questing a change in speed.

After the two programs are entered, the BASIC program is initialized. If all 42 characters are desired, the number 42 is entered on Prompt. Any speed may be entered on the speed Prompt. After 100 characters are sent, a speed Prompt will appear. A speed change may then be made which retains the original set of characters specified in either Statement 100 or 110. The number of lines to be sent may be altered by changing "If *x* = 5 then 150" to some other integer value of "*x*".

This program has been used on a regular basis for several months. People who have listened to the code lessons generated by this program have responded favorably.

We hope to modify the program to enable entering text and having it sent as Morse Code by the variable speed and space routines described above.

Bibliography

1. DeJong, Marvin L., "AIM 65 in the Ham Shack," MICRO, September 1979, (16:29).
2. DeJong, Marvin L., "AIM 65 Morse Code Send Program," MICRO, September 1979, (16:51).
3. DeJong, Marvin L., "A Complete Morse Code Send/Receive Package," MICRO, February 1980, (21:19).

MICRO

Basic Listing 1.

```
 10 DIM A(20),C(50)
 20 C(1)=44:C(2)=45:C(3)=46:C(4)=47
 30 FOR I=5 TO 16
 40 C(I)=I+43
 50 NEXT I
 60 FOR I=17 TO 42
 70 C(I)=I+48
 80 NEXT I
 90 PRINT"ENTER NUMBER OF CHARACTERS DESIRED":INPUT N
 100 IF N=42 THEN 150
 110 PRINT"ENTER CHARACTERS DESIRED":INPUT A$
 120 FOR I=1 TO N
 130 C(I)=ASC(MID$(A$,I,1))
 140 NEXT I
 150 PRINT"ENTER SPEED":INPUT S
 151 Z=S:X+0
 155 T=1:U=7500/S
 160 IF S>14 THEN 190
 170 T=7500/S-500
 180 S=15
 190 S=INT(1172/S+.5)
 200 POKE 55,S
 210 POKE 04,92
 220 POKE 05,15
 230 FOR I=1 TO 20
 240 J=INT(N*RND(1)+1)
 250 A(I)=C(J)
 260 NEXT I
 270 J=0
 280 FOR I=1 TO 20
 290 POKE 56,A(I)
 300 FOR C=1 TO T
 310 NEXT C
 320 Y=USR(0)
 330 IF I=20 THEN PRINT:GOTO 400
 350 J=J+1
 360 IF J<>5 THEN 400
 370 FOR J=1 TO U
 380 NEXT J
 390 J=0
 400 NEXT I
 410 X=X+1
 420 IF X=5 THEN 150
 430 GOTO 230
```

An Apple Flavored Lifesaver

The game of life is made a little easier with this flexible storage program which provides for translation, rotation, and reversal of patterns.

Gregory L. Tibbetts
31417 49th Place S.W.
Federal Way, WA 98003

John Conway's game of Life has one of the largest followings of any computer simulation ever devised. My own interest dates back to my first "cellular excursion" in 1972, on a Hewlitt-Packard 2000c machine, Since then I've collected half a dozen versions and have played with several more, all widely different in execution. One serious drawback nearly every version shares, however, is the sheer drudgery of entering from 2 to 200 sets of coordinates each time a simulation is to be run. I've seen several programs with systems to capture coordinates for a given figure—some plain and some incredibly complex. All of these though, are hampered by the fact that Life devotees rarely input the same pattern at exactly the same location and orientation twice, and they usually like to combine figures for interactive effects. One system attempting to circumvent these problems had over 120 individual figures on paper tape, most duplicated up to 8 times for different orientations, and all marked and cataloged. Now that's dedication!

Being basically lazy myself, (after all, I bought a computer to save myself work), I decided that I needed a few simple routines that would let me name and save figures to disk, and then call them back to the screen at virtually any location, at any reasonable orientation, and in combination with any other pattern on file. My goal then, and the subject of this article, is simply to make Life a little easier. (Pun intended.)

The platform I chose to build my routine on is an excellent machine code/Integer BASIC hybrid program written by Dick Suitor. It appeared most recently in *Best of Micro*, Volume II. Probably the best and most versatile of all the versions I have seen, it has features like variable generation speed, the ability to set random cells alive in a selected field, and the use of contrasting color to show cell development.

My first task was to come up with a method of storing and retrieving the figures. The obvious solution was to save the *x,y* coordinates in a sequential text file. In order to make the figures completely relocatable however, I needed a way to make the stored coordinates independent of the screen coordinates. The method I chose was to select an arbitrary centerpoint for the figure, prior to input. Then as each coordinate set was typed in, the *x, y* values of the center point would be subtracted from the *x, y* values of the point being entered. The result is a set of codified *x, y* values, positive and negative, which are relative only to the centerpoint, and therefore totally independent of their current screen location. All that's required to relocate the figure then, is to change the centerpoint when calling the figure back from storage.

This method, in conjunction with APPLE's system of screen coordinates, does introduce an irregularity which which will become important as we proceed. In normal coordinate systems *x* values increase as we move to the right, and *y* values increase as we go up. With the APPLE II, *y* values increase as we decend on the screen. Further, all screen coordinates are positive, while the codified values may be positive or negative, since they essentially make up a coordinate grid of their own, with the *x* (horizontal) and *y* (vertical) axes intersecting at the chosen centerpoint. Unlike normal grids, therefore, *y* values will be negative above this *x* axis and positive below it. It will be necessary to keep this in mind, as it is the codified values we will be manipulating in the coming paragraphs when we determine how to reorient the figures.

This second task, that of finding a way to bring the stored figure back to the screen in a different attitude than originally entered, was somewhat more difficult than simply making it relocatable. However, it quickly became clear that all possible orientations could be achieved by reversing the figure, rotating it, or both.

Rotation is obtained by moving each point clockwise around the center some distance (depending on the degree of rotation), while reversal takes the two dimensional image and flips it over, as one would turn over a playing card. Obviously reversal requires us to know which axis the figure is to be reversed around.

Defining an algorithm to rotate and reverse the figures was an interesting exercise, (actually three exercises and three algorithms). I'm sure that somewhere in the field of coordinate mathematics there exists specific rules for such operations. Being more a tinkerer than a scholar, however, I chose to discover those rules by trial and error. Armed with graph paper and pencil, I defined a center, and *x* and *y* axis, and began examining what happened to various sets of coordinates when the points they described were reversed or rotated. The first thing I discovered was that for any single set of coordinates, rotation or reversal involved only two operations: either the unsigned magnitudes of the *x* and *y*

values being swapped, or the signs of one or both values being changed. One, or a combination of these two alterations will produce all feasible orientations. I also learned that rotations in other than 90° increments were not feasible for the purposes of the Life game, but the proof of that is left as an exercise for the reader.

The reversal mechanism turned out to be the simplest. A little paper and pencil work showed that no matter which axis was used for reversal, any point remained the same distance from each axis when reversed. The magnitudes of the x and y values then must remain the same. The signs, however, do not. A reversal around the y axis, for example, sends points from the upper right quadrant $(+x, -y)$ to the upper left quadrant $(-x\ -y)$, and from lower right $(+x, +y)$ to lower left $(-x, +y)$. Obviously then, reversal on the y axis changes the sign of the x values only. By the same token, an x axis reversal changes the sign of the y values only. Translated into a sequence of program steps this mechanism is implemented in program lines 1070-1110 and 350-400. I also re-solved the further question of whether multiple reversals were desirable, that is, two reversals around one axis, or one around each. I determined they were not, but as a second exercise, for fun, the reader may wish to prove why they were not.

Rotation was a little harder as the cases of 90°, 180°, and 270° rotation all had to be allowed for. Easiest to discover was the 180° process. Just as in the reversal case, a point rotated 180° still remains the same distance from each axis, and therefore, the x and

BASIC Listing

```
0 LOMEM:2500
10 DIM HEX$(30)
20 HEX$="0800<94FB.85FFM NE88AG"
30 FOR I=1 TO LEN(HEX$): POKE 511+I, ASC
(HEX$(I)): NEXTI: POKE 72,0: CALL -144
40 DEL 0,40
50 GOTO 800
60 POKE -16302,0: COLOR=0: FOR K=40 TO 4
7
70 HLIN 0,39 AT K: NEXT K
80 KX= PDL (0)-10: IF KX>240 THEN KX=KX1
: IF KX<0 THEN KX=0
90 K1=KX*6:K2=KX*2:K3=500/(K1+50)+1
100 FOR I=1 TO K3
110 CALL GEN
120 FOR K=1 TO K2: NEXT K
130 CALL MOP
140 FOR K=1 TO SIZE: COLOR=11
150 NEXT I
160 GOTO 80
170 FOR I=1 TO SIZE: COLOR=11
180 X=XCTR+X(I):Y=YCTR+Y(I)
190 IF X<0 OR X>39 OR Y<0 OR Y>39 THEN 1
210
200 PLOT X,Y: NEXT I
210 RETURN
220 FOR I=I1 TO I2: FOR J=J1 TO J2
230 COLOR=11: IF RND (L) THEN COLOR=0
240 PLOT I,J
250 NEXT J: NEXT I
260 GOTO 60
270 FOR I=1 TO SIZE
280 X=Y(I):Y=X(I)
290 IF Y(I) THEN X=X*-1
300 X(I)=X:Y(I)=Y
310 NEXT I: RETURN
320 FOR I=1 TO SIZE
330 X(I)=X(I)*-1:Y(I)=Y(I)*-1
340 NEXT I: RETURN
350 FOR I=1 TO SIZE
360 IF XAX THEN 380
370 X=X(I):Y=Y(I)*-1: GOTO 390
380 Y=Y(I):X=X(I)*-1
390 X(I)=X:Y(I)=Y: NEXT I
400 RETURN
410 PRINT D$;"OPEN";A$
420 PRINT D$;"READ";A$
430 FOR I= 1 TO 255
440 INPUT X(I),Y(I)
450 IF X(I)=99 OR Y(I)=99 THEN 470
460 NEXT I
470 SIZE=I-1
480 PRINT D$;"CLOSE";A$
490 IF ROT THEN GOSUB 270
500 IF HALF THEN GOSUB 320
510 IF REV THEN GOSUB 350
520 GOSUB 170
530 HALF=0:ROT=0:REV=0:XAX=0:SIZE=0
540 RETURN
550 PRINT D$;"OPEN";A$
560 PRINT D$;"DELETE";A$
570 PRINT D$;"OPEN";A$
580 PRINT D$;"WRITE";A$
590 FOR I=1 TO SIZE
600 PRINT X(I)
610 PRINT Y(I)
620 NEXT I
630 PRINT D$;"CLOSE";A$
640 RETURN
650 FOR I=1 TO 255
660 INPUT X,Y
670 IF X=99 OR Y=99 THEN 720
680 IF X<0 OR X>39 OR Y<0 OR Y>39 THEN 7
00
690 X(I) =X-XCTR:Y(I)=Y-YCTR: GOTO 710
700 PRINT "INPUT X,Y",X,Y
710 NEXT I
720 X(I)=99:Y(I)=99
730 SIZE=I
740 RETURN
```

(continued)

```
750 INPUT "INPUT X,Y",X,Y
760 IF X=99 OR Y=99 THEN 60
770 IF X<0 OR X>39 OR Y<0 OR Y>39 THEN 7
90
780 COLOR=11: PLOT X,Y: GOTO 750
790 PRINT "OUT OF RANGE!": GOTO 750
800 TEXT
810 DIM X(255,Y(255),A$(50),B$(2),D$(1)
820 GEN=2088:MOP=2265:K1=1:K2=1:D$="": R
EM  D$=CNTRL D
830 CALL -936:VTAB 5:TAB 9:PRINT"CONWAY'
S GAME OF LIFE":FOR I=1 TO 700:NEXT I
840 GR
850 PRINT "DO YOU WISH TO: 1.PLAY OR 2.C
REATE"
860 INPUT "A NEW PATTERN FILE (1/2).",C1

870 IF C1=2 THEN 1140
880 INPUT "SPEED=PDL(0): SET DEFAULT (0-
255)",KX1
890 PRINT "DO YOU WISH TO: 1.RANDOM PATT
ERN 2. PATTERN"
900 INPUT "SPEED=PDL
900 INPUT "FROM DISK OR 3.STANDARD: (1/2
/3)",C1
910 IF C1=3 THEN 990
920 IF C1=2 THEN 1010
930 INPUT "X DIRECTION LIMITS ",I1,I2
940 IF I1<0 OR I2>39 OR I1>I2 THEN 930
950 INPUT "Y DIRECTION LIMITS ",J1,J2
960 IF J1<0 OR J2>39 OR J1>J2 THEN 950
970 INPUT "ONE IN 'N' CELLS WILL LIVE: E
NTER N",L
980 GOTO 220
990 PRINT "ENTER YOUR PATTERN (X,Y):99,9
9 EXITS"
1000 GOTO 750
1010 INPUT "WHAT FIGURE NAME",A$
1020 INPUT "ENTER CENTER COORD'S (X,Y)",
XCTR,YCTR
1030 INPUT "ENTER ROTATION (0/90/180/270
)",ROT
1040 IF ROT=180 OR ROT=270 THEN HALF=1
1050 IF ROT=90 OR ROT=270 THEN ROT=1
1060 IF ROT<>1 THEN ROT=0
1070 INPUT "ENTER 1.REVERSED OR 2.STANDA
RD (1/2)",REV
1080 IF REV>1 THEN REV=0: IF NOT REV THE
N 1110
1090 INPUT "REVERSE ON 1.X-AXIS OR 2.Y-A
XIS (1/2)",XAX
1100 IF XAX>1 THEN XAX=0
1110 GOSUB 410
1120 INPUT "ANOTHER FIGURE (Y/N)",B$: IF
 B$="N" THEN 60
1130 PRINT "CAUTION:FIGURES MAY OVERWRIT
E!": GOTO 1010
1140 INPUT "ENTER CENTER COORD'S (X,Y)",
XCTR,YCTR
1150 PRINT "ENTER ALL LIVE CELLS (X,Y):
99,99 EXITS"
1160 GOSUB 850
1170 INPUT "ENTER NAME FOR THIS FIGURE",
A$
1180 GOSUB 550
1190 PRINT "TESTING": GOSUB 410
1200 GOTO 60
1210 PRINT "PLOT ABORTED/FIGURE WENT OFF
 SCREEN"
1220 PRINT "MOVE CENTERPOINT:X AND Y WHE
N ABORTED"
1230 PRINT "WERE ";X;",";Y: POP: POP
1240 IF I=1 THEN 1020: IE=I-1: COLOR=0:
FOR I=1 TO IE
1250 PLOT X(I)+XCTR,Y(I)+YCTR: NEXT I: G
OTO 1020
1260 REM ADAPTATION BY GREG TIBBETTS OF
RICHARD SUITOR'S PROGRAM IN
1265 REM "BEST OF MICRO" VOLUME II 1979
1270 REM LINES 0-50 PROGRAM SET-UP
1280 REM 60-160 SPEED AND GENERATION
1290 REM 170-210 GENERAL PLOT SUBR.
1300 REM 220-260 RANDOM PLOT SUBR.
1310 REM 270-340 ROTATION SUBR'S.
1320 REM 350-400 REVERSAL SUBR.
1330 REM 410-540 DISK READ SUBR.
1350 REM 750-790 STANDARD INPUT SUBR
1360 REM 800-840 INITIALIZATION
1370 REM 850-920 MODE SELECTION
1380 REM 930-1200 USER INPUT/SELECT
1390 REM1210-1250 PLOT ABORT SUBR.
10000 END
144542243 LOAD 6146 HLIN 115726885
     HIMEM:I RUN P CLR 26884,I INPUT
     HIMEM: RUN 43268 LOAD 6149 IF
     (8444J RUN ^ HIMEM: RUN 24577
     389511692368241661442 MAN HIMEM:
      CON ~$ DEL ~= DEL " RUN P:
     ~$ DEL 61444 MAN HIMEM::~= DEL
     " RUN p CON ~= DEL ~$ DEL 4298
     N ,>=2407, COLOR+ DEL 44294
     2791334057 RUN >=30982( DEL
     (y( DEL 51462:p,51460 LOAD
     p,61442,"11042306 SGN 614630983
      VTAB DEL (y) DEL 51463:p,51460
      LOAD p,61444,"x TAB LOAD 2308
     :34820p _ HIMEM:11084 RUN 34052
      SAVE 34048 LOAD 24082456 IF
     ^ J RUN ^ HIMEM: RUN 2457745351
```

y magnitudes remain the same. Signs however, do not follow the same pattern as during reversal. Since the points in the upper right quadrant $(+x, -y)$ move to the lower left $(-x, +y)$, lower right$(+x, +y)$ to upper left $(-x, -y)$ and vice versa, it becomes clear that both x and y values must change sign. A 180° rotation therefore is accomplished by simply multiplying the two values by -1. This is implemented in lines 1030-1060 and 320-340.

A 90° rotation is not so straightforward. It is best seen by using the example of a clock face with the x axis running through the 9 and 3, and the y axis through the 12 and 6. A 90° rotation of this clock face moves the point at numeral 1 to the position of numeral 4. For the first time, the magnitude of the x and y values have changed. The distance of the point from the y axis in its original postion has become the distance from the x axis after rotation and vice versa. What happens in a 90° rotation then, is that the magnitudes of x and y are simply exchanged. The signs, unfortunately, do not follow such a clearcut pattern. Nevertheless, a pattern does exist. I found it by examining the four quadrants in sequence and noting what happens to their associated x and y signs. Starting at the upper right $(+x, -y)$ and moving to the lower right produces $(+x, +y)$. Another 90° rotation produces $(-x, +y)$, and the final rotation $(-x, -y)$. Study here shows that the sign of x in the original quadrant is the sign y will have in the new quadrant. Since the magnitude of x becomes the magnitude of y also, we can simply give y the signed value of x for every point to be rotated. You can also see that the sign of the new x value is the opposite of the old y value. To get the new x value we must multiply the old signed value of y by -1. These two steps complete the 90° algorithm and it is implemented in lines 1030-1060 and 270-310. To keep the program as short as possible, 270° rotations were made by using the 90° and 180° subroutines together. This completes the screen output design.

Disk storage is achieved by saving the x and y arrays into a sequential text file; each figure to a separate file. Though this is somewhat wasteful of disk space, I set it up this way to avoid complex file management routines, and to allow for easy renaming and catalog display. The final step was to insert tests in the plot sequence to prevent range errors from crashing the program if a center point was selected that would cause the figure to plot off the screen, and having to restart the program from scratch. The original centerpoint is not stored with the codified values, and consequently is not available for later examination.

Machine Code Listing

```
0800 A5 05     LBLI  LDA  $0005
0802 85 03           STA  $0003
0804 A5 04           LDA  $0004
0806 85 02           STA  $0002
0808 18              CLC
0809 69 80           ADC  #$80
080B 85 04           STA  $0004
080D A5 05           LDA  $0005
080F 69 00           ADC  #$00
0811 C9 08           CMP  #$08
0813 D0 0C           BNE  LBLA
0815 A5 04           LDA  $0004
0817 69 27           ADC  #$27
0819 C9 52           CMP  #$52
081B 10 08           BPL  LBLB
081D 85 04           STA  $0004
081F A9 04           LDA  #$0004
0821 85 05     LBLA  STA  $0005
0823 18              CLC
0824 60        LBLR  RTS
0825 38        LBLB  SEC
0826 B0 FC           BCS  LBLR
0828 20 CA 08        JSR  LBLS
082B 20 00 08  LBLX  JSR  LBLI
082E 90 01           BCC  LBLC
0830 60              RTS
0831 A0 27     LBLC  LDY  #$27
0833 98              TYA
0834 AA              TAX
0835 A9 00     LBLH  LDA  #$00
0837 99 40 09        STA  $0940,Y
083A 99 70 09        STA  $0970,Y
083D B1 02           LDA  ($02),Y
083F F0 0F           BEQ  LBLE
0841 10 0A           BPL  LBLD
0843 FE 40 09        INC  $0940,X
0846 FE 70 09        INC  $0970,X
0849 29 08           AND  #$08
084B F0 03           BEQ  LBLE
084D FE 40 09  LBLD  INC  $0940,X
0850 B1 04     LBLE  LDA  ($04),Y
0852 F0 0F           BEQ  LBLG
0854 10 03           BPL  LBLF
0856 FE 70 09        INC  $0970,X
0859 29 08     LBLF  AND  #$08
085B F0 06           BEQ  LBLG
085D FE 70 09        INC  $0970,X
0860 FE 40 09        INC  $0940,X
0863 88        LBLG  DEY
0864 CA              DEX
0865 10 CE           BPL  LBLH
0867 A0 26           LDY  #$26
0869 18              CLC
086A AD 67 09        LDA  $0967
086D 6D 66 09        ADC  $0966
0870 85 06           STA  $0006
0872 AD 97 09        LDA  $0997
0875 6D 96 09        ADC  $0996
0878 85 07           STA  $0007
087A 18        LBLW  CLC
087B A5 06           LDA  $0006
087D 79 3F 09        ADC  $093F,Y
0880 38              SEC
0881 F9 42 09        SBC  $0942,Y
```

The program as it appears in the listing, is set up to run on a 48K APPLE II, using APPLE DOS to store and retrieve the patterns. The instructions for setting up the program, however, are universal with respect to RAM size. I believe that the program could also be converted to use a cassette-based DOS imitator as off-line storage, but that is beyond the scope of this article, (and the ingenuity of the writer as well). The machine code runs resident at \$800 (2048), and the program has been modified to load both sections as a unit, and relocate the machine portion when run. To enter the program, first the machine code must be typed in (from the hex dump below), to occupy the top 261 bytes of RAM. HIMIM: and PP (program pointer) must then be moved down to protect it, the BASIC portion entered, and then HIMEM: moved back to its original value. The BASIC program is then altered to automatically move LOMEM: when run and relocate the code to this protected area. Readers familiar with these procedures may skip the instructions which follow. For the purposes of these instructions however, it is assumed that the reader is knowledgeable in the process of converting decimal to hexadecimal and back, and is familiar with APPLE's low order, high order byte storage and the method of converting this to whole number values or visa versa.

REM: Entering the Machine Code

1. Boot DOS, type INT and PEEK memory locations 76 and 77 (HIMEM:). Convert these two numbers to their whole number equivalent and that to its hex equivalent. Record all of these numbers.

2. Compute the starting address for the code by subtracting 262 from the whole number value of HIMEM:. Convert this to hex and again record the numbers.

3. Call -151 and enter the 261 bytes of code, starting at the hex address you calculated in step 2. Once entered, type (CNTRL) C and BSAVE LIFE OBJ, A\$XXXX, L261; where XXXX is the hex address from step 2.

4. Now convert the above starting address, minus 1 byte, from a decimal whole number to its low order and high order values. POKE these values as follows: POKE 76, low; POKE 77, high; POKE 202, low; and POKE 203, high. Code is now protected.

REM: Entering the Basic Program

5. Enter line 0 and line 40 as PRINT statements to avoid the SYNTAX ERROR message that would come if LOMEM: and DEL were entered as deferred commands.

6. Enter line 20 exactly as shown with two exceptions. Where the listing shows 94FB, substitute the hex value

(continued)

```
0884 85 06            STA   $0006
0886 C9 03            CMP   #$03
0888 F0 0E            BEQ   LBLK
088A 90 04            BCC   LBLJ
088C C9 04            CMP   #$04
088E F0 0E            BEQ   LBLL
0890 B1 02      LBLJ  LDA   ($02),Y
0892 F0 0A            BEQ   LBLL
0894 29 85            AND   #$85
0896 50 04            BVC   LBLM
0898 B1 02      LBLK  LDA   ($02),Y
089A 09 30            ORA   #$30
089C B1 02      LBLM  LDA   ($02),Y
089E 18         LBLL  CLC
089F A5 07            LDA   $0007
08A1 79 6F 09         ADC   $096F,Y
08A4 38               SEC
08A5 F9 72 09         SBC   $0972,Y
08A8 85 07            STA   $0007
08AA C9 03            CMP   #$03
08AC F0 0E            BEQ   LBLP
08AE 90 04            BCC   LBLN
08B0 C9 04            CMP   #$04
08B2 F0 0E            BEQ   LBLT
08B4 B1 04      LBLN  LDA   ($04),Y
08B6 F0 0A            BEQ   LBLT
08B8 29 F8            AND   #$F8
08BA 50 04            BVC   LBLV
08BC B1 04      LBLP  LDA   ($04),Y
08BE 09 03            ORA   #$03
08C0 91 04      LBLV  STA   ($04),Y
08C2 88         LBLT  DEY
08C3 F0 02            BEQ   LBLU
08C5 10 B3            BPL   LBLW
08C7 4C 2B 08   LBLU  JMP   LBLX
08CA A9 04      LBLS  LDA   #$04
08CC 85 05            STA   $0005
08CE A9 00            LDA   #$00
08D0 85 04            STA   $0004
08D2 8D 68 09         STA   $0968
08D5 8D 88 09         STA   $0988
08D8 60               RTS
08D9 20 CA 08         JSR   LBLS
08DC 20 00 08   LABD  JSR   LBLI
08DF 90 01            BCC   LBLY
08E1 60               RTS
08E2 A0 27      LBLY  LDY   #$27
08E4 B1 02      LBLO  LDA   ($02),Y
08E6 F0 0A            BEQ   LBLZ
08E8 29 7F            AND   #$7F
08EA C9 10            CMP   #$10
08EC 30 02            BMI   LABA
08EE 09 80            ORA   #$80
08F0 91 02      LABA  STA   ($02),Y
08F2 B1 04      LBLZ  LDA   ($04),Y
08F4 F0 0A            BEQ   LABB
08F6 29 F7            AND   #$F7
08F8 6A               ROR   A
08F9 90 02            BCC   LABC
08FB 09 04            ORA   #$04
08FD 2A         LABC  ROL   A
08FE 91 04            STA   ($04),Y
0900 88         LABB  DEY
0901 F0 D9            BEQ   LABD
0903 10 DF            BPL   LBLO
```

you calculated in step 2. Where it shows 95FF, substitute the original value of HIMEM: from step 1, minus 1 byte. The format of this string must be exact, as it becomes an APPLE monitor command when the program is run. Be sure the spacing etc., match. Enter all other statements normally.

7. To create the LOMEM: and DEL statements, PEEK locations 202 and 203 (PP) to find the starting address of the BASIC program. Convert these to a single hex value and Call −151. Beginning with that location, examine sequential locations until a byte 62 is found, (this should be within the first 5 bytes). This is the token for the print in line 0. Change this byte to an 11, the LOMEM: token. Keep reading sequentially until a second 62 is found and change this to 09, the DEL token. You must also change the 49 three bytes further on to 0A, changing the PRINT comma to a DEL comma.

8. Now by entering (CNTRL) C and LIST 0, 40; you should see your listing match the one in this article. For safety's sake, save the BASIC program as LIFE B.

REM: Combining the Two

9. Take the original low order and high order byte values for HIMEM: from step 1, and POKE these into locations 76 and 77, respectively. DO NOT RUN THE PROGRAM YET! Now when you SAVE LIFE, the APPLE will obediently save everything from PP to HIMEM:, tucking your machine code safely at the end of your BASIC program.

At this point the program may be run, listed and even changed without difficulty. I would suggest, however, that you keep LIFE OBJ and LIFE B until the combined program is throughly use-tested. REM lines at the end of the listing will aid trouble shooting if it becomes necesssary. The program is completely automated and self-prompting, therefore I have only a few helpful hints.

First, patterns are best developed on, and input from graph paper numbered along the top and side to match the screen. This gives a backup as well as a hard copy visual image to check the screen output. Second, the centerpoint you select to input the figure is not automatically set as a live cell. Consequently, it can literally be any point on the screen. You must remember though, that all figures are rotated and reversed around this relative center, and therefore, it should be chosen with care. Third, with really large figures where the choice of center point is critical to keep from plotting the figure off screen, it is helpful to include the center coordinates in the figure name as a guide during recall. Last, due to the finite field limits established by Mr. Suitor's program, known patterns may not behave normally if they contact the edge. Gliders for example, turn to boxes as they hit the edge, rather than continue to move off screen. This is no cause for alarm; simply a fact of Life.

For fun, create a pattern file with the coordinates listed below. Name this figure PULSAR SEED, and use an initial centerpoint of say 19,19. When you run it the results may surprise you. In any case, have fun!

(x,y); (10,8); (9,9); (11,9); (9,10); (11,10); (9,11); (10,11); (11.11); (9,12); (11,12);(9,13); (11,13); (10,14); (99,99).

Live long and prosper.

MICRO PET Vet

By Loren Wright

Commodore's recent announcement of several new products has diverted me, for the moment, from my coverage of existing products.

VIC 20
Low-Cost, Full-Feature Computer

The most exciting new product is the VIC 20, billed as "the first full-featured, expandable color computer system selling under $300." VIC 20 includes 5K of RAM and connects to any TV or monitor. To work with the 6502 in this new computer, MOS Technology has designed a special chip, which includes CRT control, RAM, and ROM. As a result, the VIC features color, sound, programmable function keys, high resolution graphics (176 x 184), and support for joysticks, paddles, and light pens. Other features include a standard-size typewriter keyboard, 23-line by 22-column display, graphics characters, and an expanded PET BASIC. Memory is expandable to 32K RAM, and provision is made for plug-in ROM cartridges, which will include a wide variety of games, educational and special-application programs. Commodore plans several peripherals for use with the VIC's serial bus (IEEE-488 is *not* supported) and for the RS-232 port. One of these, the CBM 2031 disk drive, has already been announced. Other peripherals planned are a tape cassette unit, a printer, and add-on accessories.

"User-friendly" is a theme Commodore applies to the VIC 20, and this goes especially for the documentation, as described in the company's press release:

> Commodore will provide excellent documentation for first-time users as well as software writers and computer entrepreneurs, in the form of books and manuals—some of which will be written and marketed by outside publishers, with Commodore's support.

The VIC, already on sale in Japan, has met with even greater response than expected. Production in the U.S. has begun already and display units will be distributed to dealers in January 1981. General availability should follow shortly thereafter. If you had planned your Christmas dollars for another computer, you should strongly consider waiting for the VIC 20.

CBM 2031
Single Floppy Disk Drive

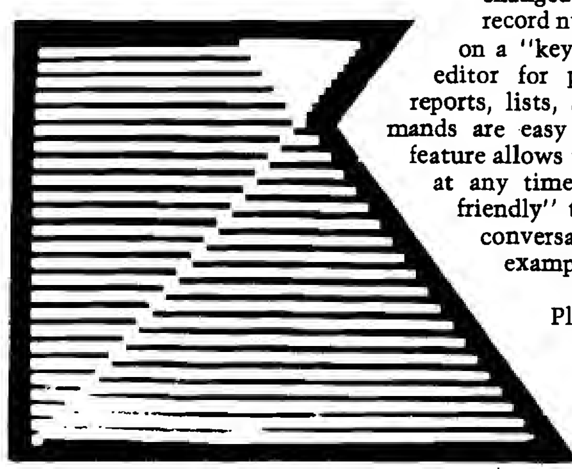
The CBM 2031 will be a low-priced single-drive floppy disk unit, available in both IEEE-4888 and serial versions. Serial-written disks will be readable on parallel disk drives, and *vice versa*, barring incompatibility in the programs themselves (such as trying color commands on a PET). Additional 2031's or dual-drive units may be added later, as needs and budgets dictate. The cost of the 2031 will be under $600.

CBM 8061/8062
"IBM-compatible" 8-inch Floppy Disk Drive

The CBM 8061 and 8062 are new "IBM-compatible" 8-inch floppy disk-drive units. ("IBM-compatible" means that the IBM 3740 data exchange format is followed.) The 8061 can handle 1.6 megabytes, using a single side of each diskette, and the 8062 can write 3.2 megabytes, using both sides of the two diskettes.

Programs written on a PET or CBM can be transferred easily to a larger computer system, and, of course, the opposite is true. At the same time, compatibility with Commodore's smaller disk-drive products is maintained. Translation through the CPU is easily accomplished with the appropriate programs. These will be available in early 1981.

80 x 50 PET Graphics



(See Davis article, this issue, p. 15)

CBM 8096
96K Business Computer

The CBM 8096, containing 96K of user RAM, has been added to the business line. This is essentially an 8032 with an added 64K RAM. Large, sophisticated programs will be usable, including the interesting possibilities of running FORTRAN and COBOL programs. CBM 8096 is available now.

Wordcraft 80
Word Processor for CBM 8032

Two new business software packages—OZZ and Wordcraft 80—were announced. Both were demonstrated in prototype versions at the National Small Computer Show in New York, which I attended on November 1. Wordcraft 80 is a powerful word-processor program designed specifically for the CBM 8032. Combined with the 8032, a disk-drive unit, and a letter-quality printer, the cost of a Wordcraft 80 system is about $5000. This compares very favorably with some dedicated systems which cost a lot more. Some of its features include merging from disk files for form letters, automatic centering and right margin justification, transfer of text from one page to another, character string search and replace, and automatic underlining and emboldening of text. Wordcraft 80 is available now.

OZZ—The Information Wizard

"OZZ—The Information Wizard" was also designed specifically for the 8032. It is a very flexible machine code program that allows the user to set up and format information on the screen. Boxes are labeled, and then may be combined in user-specified calculations to update the contents of other boxes. These calculations may also be easily changed. Access to disk files is by record number, index title, or by search on a "key word." There is a document editor for printed information such as reports, lists, and mailing labels. The commands are easy to remember, but a "help" feature allows the user to refresh his memory at any time. Commodore applies "user-friendly" to OZZ's documentation. It is conversationally written, with many examples. OZZ is available now.

Please address correspondence to: MICRO
Pet Vet
P.O. Box 6502
Chelmsford, MA 01824

MICRO Classified

OSI Tee-Shirts
Black Mens shirt with blue on white OSI Logo. We're the only source! Send size and $7.50 plus $.75 postage per shirt to: COMPUTER BUSINESS SERVICE, P.O. Box 20384, San Jose, CA 95120.

PROGRAMMER FATIGUE?
SYM-BUG/MONEX adds 15 commands to SYM's repertoire including an interactive trace/debug. Cassette at $0200 or $3800: $19.95. EPROM (2716-5V) at $F000-$F7FF: $39.95. Commented source listing: $9.95. RAE-1(1/2) FORMAT CASSETTE: $35 (requires 8K). Custom assembly add $2.00. Foreign add $2.00. SASE for more information. Jeff Holtzman, 6820 Delmar-203, St. Louis, MO 63130.

C1P EXTENDED MONITOR
2K EPROM has 14 cursor control/editing functions, improved keyboard decoding. Machine lang. save, load, display, modify, move, breakpoint processing and much more. For 24, 32, 64 char/line. $39.95 plus $1.00 shipping. $1.00 for complete info. BUSTEK, P.O. BOX A, St. Charles, MO 63301.

Agenda · Memory Aid
16K IB program with options: New tasks, Read and Write database using tape or diskette, Agenda listing, Enter or Log tasks done, Daily or Total score, Correct data, and Sort tasks. Tape is $7.50, diskette $10.00. Full documentation included. Microspan Software, 709 Caldwell St., Yoakum, TX 77995.

SYM-1 Books by Robert Peck
Monitor Theory Manual $8.00. Hardware Theory Manual $6.00. SYM/KIM Appendix to First Book of KIM $4.25. Send SASE for details. Datapath, P.O. Box 2231, Sunnyvale, CA 94087.

Integer Pascal System
for the APPLE II. Compiler, interpreter and translator included for $65.00. Produces 6502 code programs for high speed. 48K and Disk required. Send for free information. M & M Software, 380 N. Armando 2-19, Anaheim, CA 92806.

—MICRO Classifieds—

A classified ad in MICRO will bring your products/services to the attention of thousands of readers. These ads are placed in clusters throughout the magazine. Each classified ad costs only $10.00 per insertion. Please limit these to no more than 40 words. Title line, name and address are not considered in the count. Ads must be pre-paid and received before the end of the month preceding the month of publication. Ads received later than the required date will be placed in the next issue. Send to:

MICRO—Classifieds
P.O. Box 6502
Chelmsford, MA 01824

NIKROM TECHNICAL PRODUCTS PRESENTS A DIAGNOSTIC PACKAGE FOR THE APPLE II AND APPLE II+ COMPUTER.

"THE BRAIN SURGEON"

All major computer systems are checked for functional hardware analysis on a regular basis for logical as well as some practical reasons. Finding what is exactly wrong can account for most of the money consuming down-time.

Apple Computer Co. has provided you with the best equipment available to date. The Diagnostic's Package was designed to check every major area of your computer, detect errors, and report any malfunctions. ***The Brain Surgeon*** will put your system through exhaustive, thorough procedures, testing and reporting all findings.

The Tests Include:

- MOTHERBOARD ROM TEST FOR BOTH APPLE II AND APPLE II+
- APPLESOFT CARD TEST
- INTEGER CARD TEST
- MEMORY RAM TEST
- DISK DRIVE ANALYSIS
- MONITOR ALIGNMENT
- DC HAYES MICRODODEM II TEST

System Diagnosis is an invaluable aid to your program library even if your system is working fine. Hours have been wasted trying to track down a "program bug" when actually hardware could be the blame!

The Brain Surgeon allows you to be confident of your system. This can be critical when file handling, sorts or backups are involved. You *must* depend on your computer during all these critical times. Running ***The Brain Surgeon*** prior to these important functions helps to insure that your system is operating at peak performance.

The Brain Surgeon is easy to use and supplied on diskette with complete documentation.

PRICE: $39.95
REQUIRES: 32 or 48K
APPLESOFT in ROM, 1 Disk Drive
DOS 3.2 or 3.3

Nikrom Technical Products

25 PROSPECT STREET • LEOMINSTER, MA 01453

Creating An Applesoft BASIC Subroutine Library

There's more than one way to run a BASIC program on your APPLE with DOS. Using EXEC files offers increased flexibility over the RUN command. In this article the author uses the power of the EXEC command to link Applesoft programs from a common library of disk-resident subroutines.

N.R. McBurney
2561 Stockbridge Rd.
Marietta, GA 30062

DISK FULL! Well, of course it was full. I had over a dozen lengthy programs stored on it. In each of those programs over fifty percent of the code was identical BASIC routines. Besides the problem of disk space, maintaining identical copies of software is almost impossible. After any given period of time identical software will differ. This is a corollary to somebody's DP axiom that "identical data bases aren't".

The first problem is to find a way to append the subroutines to the main programs. To do this, we need to know how BASIC programs are stored in RAM. With ROM BASIC, the user program usually starts in location 2049 ($801). RAM (cassette) BASIC normally starts at 12289 ($3001). All of the examples in this article assume and were executed with the ROM version of Applesoft BASIC. This start address is stored in locations 103-104 ($67-$68). Similarly, the end of the program is pointed to by locations 175-176 ($AF-$B0). This is shown graphically in figure 1, step 1. If we change the start of the program pointer to the end of our program (figure 1, step 2), then load our subroutines (figure 1, step 3) we need only change our start of program pointer back to its original value (figure 1, step 4), to to have successfully merged our two programs. To do this manually, first:

LOAD MAIN PROGRAM

where MAIN PROGRAM is the name of the file containing your BASIC program minus your subroutines. Next type:

I = PEEK(176)★256 + PEEK(175) − 2

As we stated before, decimal locations 176-175 ($AF-$B0) contain the address of the end of the program currently in RAM. Now type:

POKE 104,INT(I/256)
POKE 103,I-INT(I/256)★256

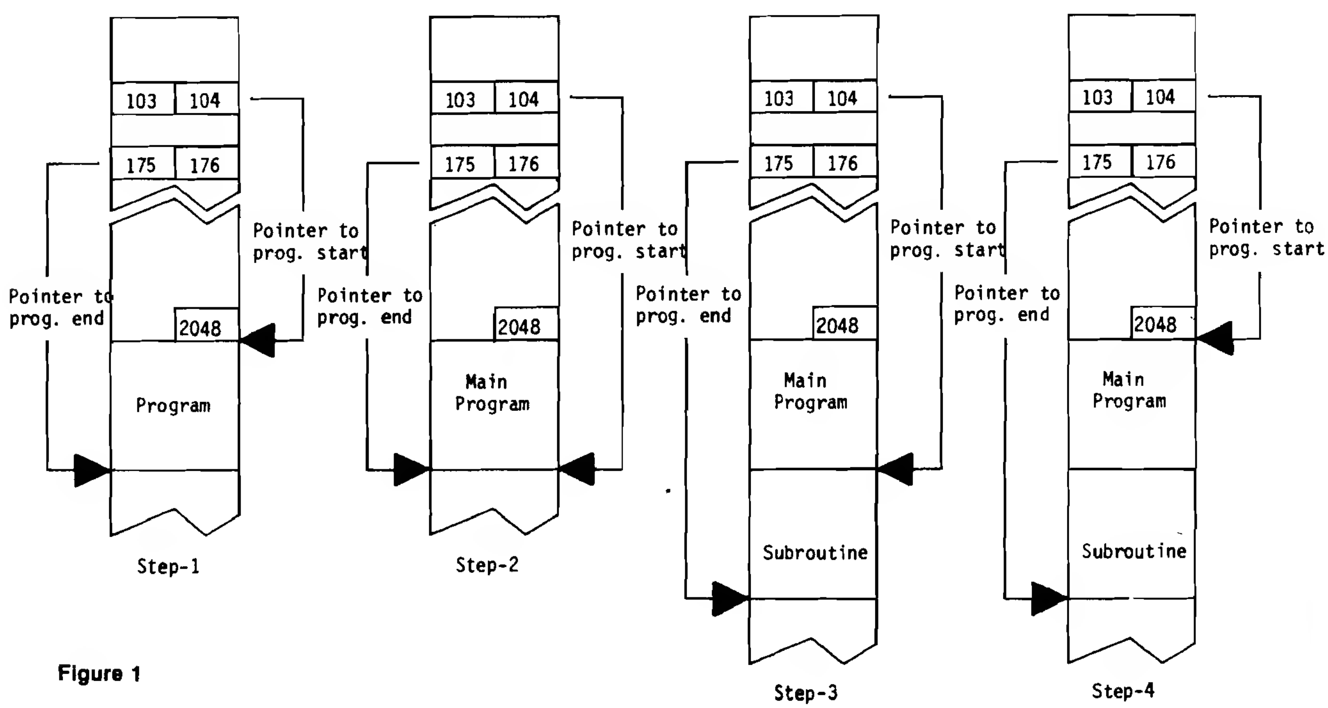


Figure 1

Decimal locations 103-104 contain the address of the start of the BASIC program. This is normally 2049 ($801). The above two statements changed the starting address to now point to the end of our main program. Now we type:

```
NEW
LOAD SUBROUTINES
```

where SUBROUTINES is the name of the file containing the routines required by MAIN PROGRAM. SUBROUTINES has now been loaded behind our first program, leaving our original program still intact in RAM. Finally we type in:

```
POKE 103,1
POKE 104,8
```

These two statements changed the pointer to the start of our program back to its original value (2049 decimal, $801 hex). Assumming that you haven't made any typing errors, if you now type LIST, you will see that you have successfully appended SUBROUTINES to MAIN PROGRAM.

When we have many programs using the same set of subroutines, this process will save core, but it doesn't result in user-oriented software. There's an easier way! The process can be handled with an EXEC file. APPLE DOS EXEC files allow you to store on a text file what you would normally type in at the keyboard. When you EXEC a file, APPLE DOS processes each line exactly as if it had been typed in at the keyboard. This is an extremely powerful tool. This article explores just one use of that power.

Program listing 1 contains code for a generalized EXEC file writer. It requests an EXEC file name, creates or replaces the file and then writes the quoted lines contained in the program's DATA statements onto the named text file. Any apostrophes (') in the DATA statements are converted to quotes (") before writing to the EXEC file. This feature allows us to write PRINT statements to the EXEC file.

If we run the program shown in listing 1, we produce the EXEC file shown in listing 2. Let's look at a simple example of how to use this EXEC file.

For this example, the file called MAIN PROGRAM (our main program) contains the instructions shown in listing 5. Our subroutine file, SUBROUTINES, contains the instructions shown in listing 6. If we type EXEC MERGE (the name of the EXEC file in listing 2), we would have the following (user input is underlined):

```
]LOAD MAIN PROGRAM
]EXEC MERGE
]

]
]
SUBROUTINES LOADED....
]LIST
      (listing appears)
```

What we've just done is create a library routine loader! While this approach has proven adequate for development work, expecting an end user to remember which main program must be used with which EXEC file is expecting a human being to adapt to the requirements of the computer. Unfortunately, this kind of design mentality has been prevalent in the industry and is responsible for much of the public's distaste for computers. A more professional approach is possible.

There are several ways that the linking operation can be made invisible to the user and more production oriented for the developer. Our previous example could have included both the LOAD MAIN PROGRAM and RUN statements. Listing 3 contains an EXEC file with these changes. Using this EXEC file results in the following:

```
]EXEC TITLE DEMO
]
]
     (Apple HOME's)
      FIRST LINE OF TITLE
       SECOND LINE
]
```

The problem with this approach is that it requires a separate EXEC file to execute each program. Every disk file requires a minimum of one sector of overhead plus one sector minimum for the program. This approach is not completely compatible with our original goal of minimizing storage requirements. A better approach, in my opinion, is to write a menu program that (invisible to the user) determines the names of the programs to be linked together by our EXEC file. These file names are stored by the menu program in RAM, and then the linking EXEC file is EXECed under program control. The EXEC file retrieves the names from RAM and runs the combined program. Listing 7 contains a sample menu program that illustrates this concept.

In our menu, program lines 1000-1190 display the menu shown in figure 2. Lines 1200-1340 request the user to enter the number of his request (the line ENTER YOUR REQUEST NUMBER... is "crawled" along the bottom of the screen). Line 1290 checks to see if a key has been

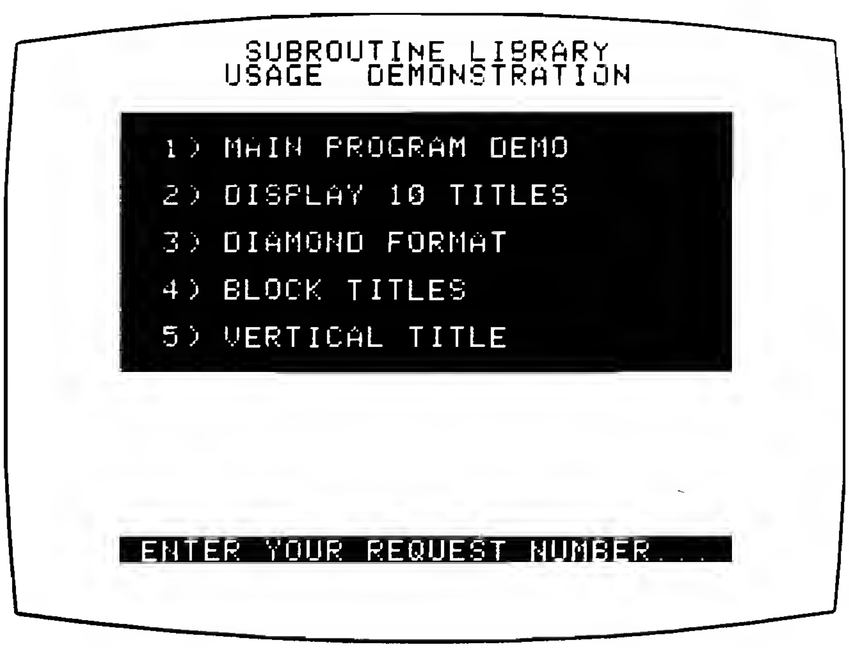


Figure 2

depressed, and if it has, line 1340 converts it from ASCII code to a digit. Lines 1350-1450 map the request number into a main program name. Since all of the programs require the same subroutine file, the name of that file is set in line 1530. The loop in lines 1550-1580 POKEs the two file names into locations 768-829 ($330-$333D). Locations 768-829 are generally available to the user. Finally, line 1610 EXECs the file MASTER MERGE shown in listing 4 and runs the desired combined programs.

I have been using various permutations of the techniques described in this article for several months and have found them to be extremely workable. The only obvious restriction is that subroutine line numbers must be larger than the last line of the main program. In practice, I've limited my main programs to lines 1-29999 and my subroutines to lines 30000-65000. The small amount of discipline that this restriction imposes is more than offset by the twin benefits of more effective disk space utilization and easier software maintenance.

```
1020 REM QUOTES IN THE DATA STATEMENTS
1021 REM ARE USED ONLY AS DELIMITERS AND
1030 REM WILL NOT APPEAR ON THE EXEC
1031 REM FILE. APOSTROPHES IN THE DATA
1040 REM STATEMENTS WILL APPEAR ASQUO-
1050 REM TATION MARKS IN THE EXEC FILE.
1060 REM
1070 D$ = CHR$ (4)
1080 HOME: PRINT CHR$(7)
1090 INPUT "NAME FOR EXEC FILE?";FILE$
1100 HOME
1110 PRINT D$;"MON O"
1120 PRINT D$;"OPEN";FILE$
1130 PRINT D$:"DELETE";FILE$
1140 PRINT D$;"OPEN";FILE$
1150 PRINT D$;"WRITE";FILE$
1160 ONERR  GOTO 1250
1162 REM  READ IN LINE AND REPLACE
1163 REM  APOSTROPHES WITH QUOTES
1164 REM
1170 READ S$
1180 A$ = ""
1190 FOR I = 1 TO LEN (S$)
1200 IF MID$(S$,I,1) <> "'" THEN A$ = A$
 + MID$(S$,I,1)
1210 IF MID$(S$,I,1) = "'" THEN A$ = A$
+ CHR$(34)
1220 NEXT
1230 PRINT A$
1240 GOTO 1170
1241 REM
1245 REM  CHECK FOR CORRECT ERROR CODE
1246 REM  (#42=OUT OF DATA)
1247 REM
1250 IF PEEK (222) = 42 THEN GOTO 1255
1251 PRINT "ERROR #"; PEEK (222)
1252 PRINT "IN LINE #" + PEEK(218) + PEE
K(219) * 2556
1253 STOP
1255 POKE 216,0
1259 PRINT D$;"NOMON O"
1260 PRINT D$;"CLOSE";FILE$
1270 REM
1280 REM  BEGIN DATA STATEMENTS DEFINING
1290 REM TEXT TO BE PLACED IN EXEC FILE
1300 REM
1310 DATA"I=PEEK(176)*256+PEEK(175)-2:PO
KE104,INT(I/256):POKE103,I-INT(I/256)*
256"
1320 DATA "LOAD SUBROUTINES"
1330 DATA "POKE 103,1:POKE 104,8:PRINT '
SUBROUTINES LOADED...';CHR$(7)"
```

Listing 1: Generalized EXEC File Writer Program

```
LOAD MAIN PROGRAM
HOME:I=PEEK(176)*256+PEEK(175)-2:POKE 10
4,INT(I/256):POKE 103,I-INT(I/256)*256
LOAD SUBROUTINES
HOME:POKE 103,1:POKE 104,8:PRINT "SUBROU
TINES LOADED...";CHR$(7)
RUN
```

Listing 2: EXEC File MERGE

```
LOAD MAIN PROGRAM
HOME:I=PEEK(176)*256+PEEK(175)-2:POKE 10
4,INT(I/256):POKE 103,I-INT(I/256)*256
LOAD SUBROUTINES
HOME:POKE 103,I:POKE 104,8
RUN
```

Listing 3: EXEC File TITLE DEMO

```
MAIN$="":FORI=1TO30:MAIN$=MAIN$+CHR$(PEE
K(767+I)):NEXT:PRINT CHR$(4);"LOAD ";MAI
N$
I=PEEK(176)*256+PEEK(175)-2:POKE 104,INT
(I/256):POKE 103,I-INT(I/256)*256
SUBR$="":FORI=1TO30:SUBR$=SUBR$+CHR$(PEE
K(798+I)):NEXT:PRINT CHR$(4);"LOAD ";SUB
R$
POKE 103,I:POKE 104,8
RUN
```

Listing 4: EXEC File MASTER MERGE

(continued)

```
100 REM
110 REM  DEMONSTRATION MAIN PROGRAM
120 REM
130 HOME
140 TITLE$ = "FIRST LINE OF TITLE"
150 GOSUB 10000
160 TITLE$ = "SECOND LINE"
170 GOSUB 10000
180 END
```

Listing 5: MAIN Program

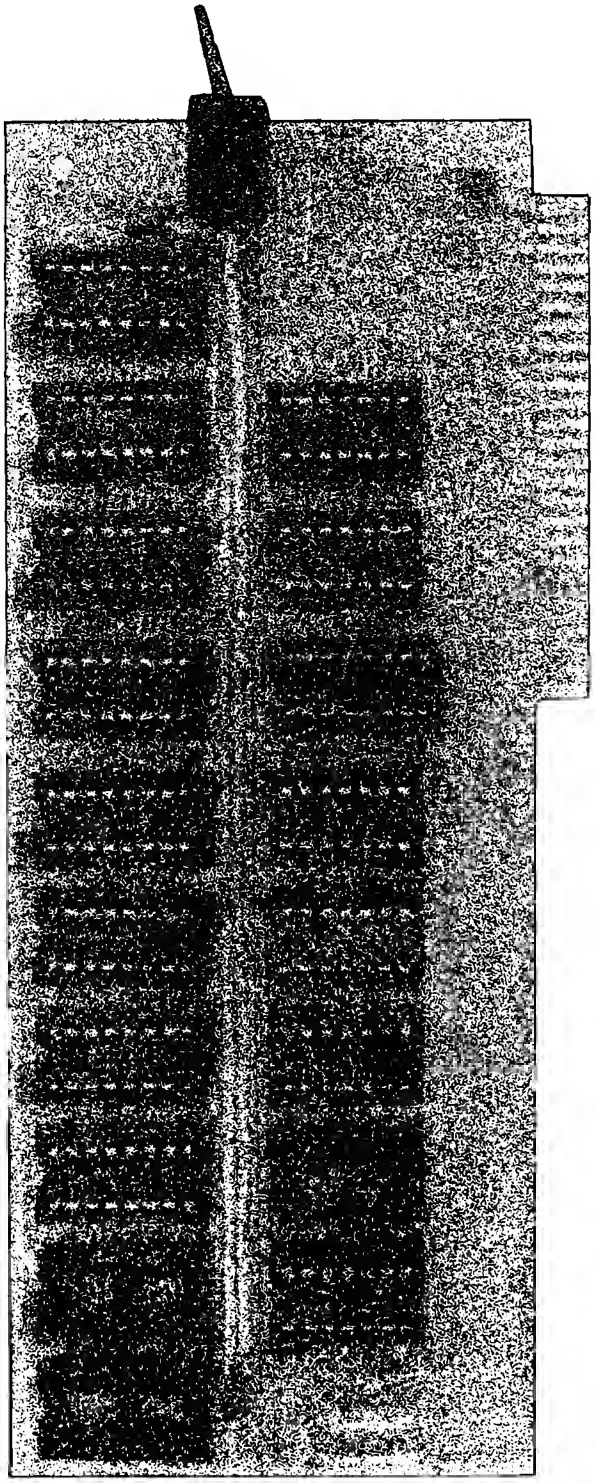
```
10000 REM
10010 REM  DEMONSTRATION SUBROUTINE TO
10020 REM  PRINT A CENTERED TITLE LINE
10030 REM
10040 L = LEN (TITLE$)
10050 PRINT TAB (20 - L / 2);TITLE$
10060 RETURN
```

Listing 6: SUBROUTINE File

```
1010 REM  MENU DEMONSTRATION PROGRAM
1020 REM
1030 HOME
1040 REM
1050 REM  DISPLAY THE MENU
1060 REM
1070 PRINT TAB( 10);"SUBROUTINE LIBRARY"
1080 PRINT TAB( 9);"USAGE DEMONSTRATION"
1090 INVERSE
1100 FOR I = 4 TO 14
1110 HTAB 5
1120 VTAB I
1130 PRINT TAB( 35)
1140 NEXT
1150 VTAB 5: HTAB 7: PRINT "1) MAIN PROG
RAM DEMO"
1160 VTAB 7: HTAB 7: PRINT "2) DISPLAY 1
0 TITLES"
1170 VTAB 9: HTAB 7: PRINT "3) DIAMOND F
ORMAT"
1180 VTAB 11: HTAB 7: PRINT "4) BLOCK TI
TLES"
1190 VTAB 13: HTAB 7: PRINT "5) VERTICAL
 TITLE"
1200 REM
1210 REM  REQUEST AND WAIT FOR INPUT
1220 REM
1230 A$+" ENTER YOUR REQUEST NUMBER..."
1240 VTAB 22
1250 HTAB 5
1260 A$ + MID$ (A$,2) + LEFT$ (A$,1)
1270 PRINT A$
1280 FOR I = 1 TO 8
1290 X = PEEK ( - 16384)
1300 IF X> 128 THEN 1330
1310 NEXT
1320 GOTO 1240
1330 POKE - 16368,0
1340 X = X - 176
1350 REM
1360 REM  DETERMINE WHICH PROGRAM TO
1370 REM  APPEND SUBROUTINES TO AND
1380 REM  THEN RUN THAT PROGRAM VIA
1385 REM  THE EXEC FILE
1390 REM
1400 IF X = 1 THEN MAIN$="MAIN PROGRAM"
"
1410 IF X = 2 THEN MAIN$ = "TEN TITLES"
1420 IF X = 3 THEN MAIN$ = "DIAMOND"
1430 IF X = 4 THEN MAIN$="BLOCK TITLES"
1440 IF X = 5 THEN MAIN$ = "VERTICAL TIT
LE"
1450 IF MAIN$ = "" THEN PRINT  CHR$(7):
GOTO 1240
1460 NORMAL
1470 REM
1480 REM  POKE NAME OF MAIN PROGRAM INTO
1485 REM  LOCATIONS $300-$31E AND NAME
1490 REM  OF SUBROUTINE FILE INTO
1500 REM LOCATIONS $31F-$33D
1510 REM
1520 K1 = 767:K2 = 798
1530 SUBR$ = "SUBROUTINES             "
1540 MAIN$ = LEFT$ (MAIN$ + "
                         ",30)
1550 FOR I = 1 TO 30
1560 POKE K1 + I, ASC (MID$(MAIN$,I,1))
1570 POKE K2 + I, ASC (MID$(SUBR$,I,1))
1580 NEXT
1590 HOME
1600 PRINT CHR$(7)
1610 PRINT CHR$ (4);"EXEC MASTER MERGE"
```

Listing 7: MENU Program

MICRO



PBASIC-DS VERSION TWO™

1. Microcomputers which can use product: PBASIC-DS VERSION TWO was written to be used with all Apple II Computers.

2. System hardware requirements: The Apple II Computer should have either 32K or 48K of installed RAM. One disk is required. The program will output to a printer if desired. Thus, a printer is an optional item.

3. System software requirements: The Apple II Computer should have Integer BASIC and also Applesoft floating point BASIC. The floating point BASIC may be either in the RAM or ROM version.

4. Product features: PBASIC-DS makes possible the use of structured programming techniques. Structured programming permits the development of large BASIC programs in either the integer or floating point version. It provides four (4) special logic features which facilitate structured programming: (a) Two Way Conditional Test; (b) Multi-Way Conditional Test; (c) Loops — I. Test Before Loop and II. Test After Loop; (d) Subroutines.

5. Product performance: The system runs well. The disk contains two (2) programs. On command, they both compile, load and run, producing good results. The user is kept advised of the program actions.

6. Product quality: The disk and manual are of good quality with timely interactive advice.

7. Product limitations: The PBASIC system does much that was not possible in BASIC. For what it does, there does not seem to be any shortcoming.

8. Product documentation: The documentation is printed and bound. The material is well organized and easily readable. The manual contains 17 pages consisting of summaries, explanation of principal features and appendices. The manual explains the concepts of structured programming and provides diagrams to characterize important logic mechanisms. Five (5) figures show representative program action. The disk contains two (2) programs to illustrate special PBASIC facilities, "Quicksort" and "Even".

9. Special user requirements: If the user has had any prior experience with FORTRAN or PASCAL he can readily appreciate the power and desirability of structured programming. Still, no supplementary langauge familiarity is required. The principles involved are well explained in the manual.

The compiler action of PBASIC is very welcome. The organization of a new program is well tested before an attempt is made to execute the program with data.

10. Price/feature/quality evaluation: PBASIC is reasonably priced ($35.00). The manual, disk and mailing obviously represent some real costs. Apparently the price of PBASIC is based on the expected distribution of several hundred copies.

11. Additional comments: Anybody who has tried to convert a simple FORTRAN program to run in BASIC can appreciate the need for PBASIC-DS. This system gives you several sorely needed logical manipulations; it cleans up and indents the listings and it compiles the program into a RUN-ready BASIC program. This system is in its second offering. Where possible, the editing and compiling are done in RAM, greatly hastening the compile and loading process. Being a second generation system shows up in many of the convenience features that are implemented in the Command Glossary provided to actuate the system.

Systems like PBASIC-DS can do much to improve the professional organization of hobbyist programming efforts.

12. Reviewer: Gordon Thompson, P.E., 724 Kewanna Avenue, Pittsburgh, PA 15234.

13. Manufacturer: Decision Systems, P.O. Box 13006, Denton, TX 76302.

STUFFIT: A TIME SAVING UTILITY PROGRAM FOR PET BASIC FILES

Use this data-stuffing program on your PET to create a large file system or to expand your present data base—without going to a second cassette.

Roger C. Crites
11880 Rio Grande
St. Louis, MO 63138

Stand-alone data bases, for such applications as recipe files or phone directories, are often based on packed data statements. The preparation, or extension, of such data statements can be expedited with STUFFIT—the data stuffing program. Incorporated within a file program, STUFFIT takes over when you wish to build, or expand the data base.

Upon entry, STUFFIT displays "DATA ENTRY MODE" in reverse field, and the number of bytes of RAM free. A nonblinking cursor appears about halfway down the screen and data to be stored is simply typed in. The delete key may be used in the usual way to correct errors. When the return key is pressed, STUFFIT assigns a line number, applies the "DATA" key word, and programs the new data statement. The number of bytes free is updated, to keep you informed of the remaining memory space. If the line exceeds the 80-character input buffer allowed by PET, STUFFIT supplies an automatic return, and adds an asterisk to the data statement to indicate a continuation to the next data line number. The data line number and basic key word are applied and the new data statement is stored in a completely transparent fashion. The operator simply types in the information he wishes to be stored.

Line 410 clears the screen, displays the amount of free memory, and positions the cursor. Line 415 turns the video on if it is off. Line 420 gets a character from the keyboard. This character is tested to see if it is a carriage return or an escape (lines 430 and 435). If it is neither, it is added to the input string. The length of the input string is checked to make sure it will not exceed the 80-character limit. Line 450 outputs the current line to the screen and loops back to get the next character. If the input character is a return, the program jumps to line 460, which reads the last data line number and increments it. Line 465 shuts off the video, for appearances, and line 470 positions and prints the input data string on the now dark screen. Line 480 prints the new line number in a data statement. Line 490 prints a direct GOTO command and repositions the cursor so that when the end statement (line 510) is executed, and the program stops, the cursor will lie on the new data line printed by line 470.

Line 500 performs the magic. It tells the PET that the return key has been hit three times. Since the program has stopped, the PET is now in the direct command mode. The screen is blank but the data statement containing the current line number, the line of data to be stored, and a direct GOTO have been printed on the screen. The PET thinks that these lines have been punched in by hand, and the screen editor dutifully passes these lines to BASIC, which modifies the program accordingly. The current line number and the new data are thereby automatically programmed. The direct GOTO is executed, getting STUFFIT running again, collecting input for the next cycle.

On a stand-alone basis, STUFFIT is useful for loading data statements in non-file programs. When writing any program that requires extensive data, I load STUFFIT and use it to enter the required data. I then erase STUFFIT from memory, leaving the data statements, and proceed with the rest of the program. STUFFIT could, of course, be modified so that when the escape character is hit, it erases itself. However, it is sufficiently short so that manually punching in its line numbers is not very time consuming.

Although very useful in this mode, STUFFIT was written to provide a flexible data entry module for use within BASIC file routines. Specifically, it is intended for use where fast bulk storage disks are not available. It enables the user to rapidly enter new data, and then return to search, sort, or process functions without stopping the program. Upon filing additional data, the program is simply saved on cassette tape. Since STUFFIT automatically appends the new information as data statements, full file support is maintained with a single cassette deck.

To illustrate the symbiotic relationship between STUFFIT and a file program, consider the following—PERSONAL DIRECTORY. This program was written to provide a business file, containing various subcontractors and professional services. It should serve to illustrate the fundamental process, and could easily be modified to provide a custom stand-alone data file for small systems without disk support.

This directory program will search by name or occupation. Under occupation, any additional pertinent information (up to 256 characters) may be stored and retrieved. Searches are based on left-justified string comparison, that is, a search for SM would find Smith, Smothers, and anyone else with a name starting with an SM. A search under occupation for PL would turn up Plumber, Plastic Dealers, etc. Of course, a search for Plumber would only find Plumbers, as a name search for Smith--Joe would only find Joe Smith.

When run, the program asks you to choose between data entry mode and search mode. A backslash is the escape character for either mode. If data entry is selected, the program jumps to STUFFIT (at line 400). To facilitate editing filed information, STUFFIT is slightly modified to add the data line number of each data statement to the information stored in that statement. This is done transparently. The only impact to the programmer is to reduce the number of characters that may be entered before an automatic return is executed. Names, phone numbers, occupations, addresses, and miscellaneous information are simply typed in. STUFFIT functions as previously described, appending the new information as data statements. When the new information is filed, an escape (backslash) returns the program to the mode selection sequence. If the search mode is selected, the program jumps to line 200. This routine asks you to select a name search or an occupation search. Upon receiving the desired name or occupation, the program jumps to a search subroutine at line 1000. This short subroutine performs the actual search. It also checks for the presence of an auto-return flag (asterisk) and joins sequential data statements as necessary, to reproduce the original data string.

The results of the search are passed back to the main routine for display. Upon reaching the end of the file, the program pauses to allow the displayed data to be used. At this point, hitting any key returns the program to the start of the search routine. A backslash will terminate the search mode and return the program to the mode selection sequence. A backslash in the mode selection sequence will stop the program.

If the amount of information to be filed is large, it may be necessary to create two or more volumes: separate tapes. For instance, a name file could be broken into Volume 1, A through M, and Volume 2, N through Z. The amount of information that can be stored in one volume depends entirely on available memory.

STUFFIT and packed data statements allow the user of a PET (without disk drive or second cassette) to achieve much of the utility of larger disk-based systems—at least insofar as phone directories, recipe files, book index files, etc. are concerned. STUFFIT also presents and illustrates a method of altering a program interactively while it is running.

Listing 1

```
5 REM***** PERSONAL DIRECTORY *****
10 ? "cs"TAB(10)"rvPERSONAL DIRECTORY"
20 ?:?:? TAB(9)"OP MODES & ENTRY CODES"
30 ? TAB(9)"————————————————————"
40 ? TAB(9)"SEARCH & DISPLAY";TAB(29)"S"
:?
60 ? TAB(9)"ENTER NEW DATA";TAB(29)"I":?
70 ? TAB(9)"ESCAPE(ANY MODE)";TAB(29)"\"
:?
80 ?:?:?"MODE CODE=?"
90 GET A$:IF A$="" GOTO 230
100 IF A$="S" GOTO 200
120 IF A$="I" GOTO 370
130 IF A$="\" THEN END
140 GOTO 90
200 REM*** SEARCH & DISPLAY ***
210 ?"csrvSEARCH & DISPLAY":?:?:RESTORE:
READ LL
220 ?"SEARCH BY NAME(N) OR OCCUPATION (O
)?"
230 GET A$: IF A$="" GOTO 230
235 IF A$="\" THEN RUN
240 IF A$="N" GOTO 270
250 IF A$="O" GOTO 280
260 GOTO 230
270 ?:?:INPUT"NAME PLEASE";M$:GOTO 290
280 ?:?:INPUT"OCCUPATION";M$
290 GOSUB 1000:REM*** GO SEARCH ***
300 IF F%<>1 GOTO 325
310 ?:? N$;TAB(23)"PHONE"P$
320 ? O$:PRINT"FILE LINE NO."L$:?
325 IF FE%=1 THEN ?"rvEND OF FILE":GOTO3
40
330 ?:?:GOTO290:REM* CONTINUE SEARCH *
350 GET A$: IF A$="" THEN 350
360 GOTO200
370 ?"csrvDATA ENTRY MODE":?:?
380 ?"DATA MUST BE ENTERED IN THREE FIEL
DS:"
386 ?:?"FIRST- NAME(LAST NAME FIRST)"
388 ?"SECOND- PHONE NUMBER"
390 ?THIRD- OCCUPATION & MISC. INFO."
392 ?:?:?:?"FIELDS MUST BE SEPARATED WIT
H A COMMA"
394 ?:?"NO COMMA'S ALLOWED WITHIN A FIEL
D"
398 GET A$: IF A$="" GOTO398
400 REM*****    STUFFIT      *****
410 ?"csrvDATA ENTRY MODEof -"FRE(0)"BYT
ES FREE":?"ch[10cd]":?"_"
415 POKE 59409,60
420 GET A$: IF A$="" GOTO 420
```

(Continued)

(Listing 1 continued)

```
430 IF A$=CHR$(13) GOTO460
435 IF A$="\" THEN RUN
440 R$=R$+A$:IF LEN(R$)<59 GOTO 450
445 R$=R$+"*":GOTO 460
450 ?"ch[10cd]":?R$"_":GOTO 420
460 RESTORE:READ N:N=N+10
465 POKE 59409,52
470 ?"chcdcdcd"N"DATA"N","R$
480 ?"10000 DATA"N
490 ?"GOTO 410":?"ch"
500 POKE525,3:POKE527,13:POKE528,13:POKE
529,13
510 END
1000 REM*** SEARCH ROUTINE ***
1010 READ L$:REM GET LINE NO.
1015 F%=0:FE%=0:IFVAL(L$)=LL THEN FE%=1
1020 READ N$,P$,O$
1030 IF RIGHT$(O$,1)="*" GOTO1080
1040 IF A$="N" AND M$=LEFT$(N$,LEN(M$))
THEN F%=1:RETURN
1050 IF A$="O" AND M$=LEFT$(O$,LEN(M$))
THEN F%=1:RETURN
1060 IFVAL(L$)=LL THEN F%=0:FE%=1:RETURN
1090 READ C$:O$=LEFT$(O$,LEN(O$)-1)+C$
1100 GOTO 1030
10000 DATA 10000
```

These programs are written for the 2.0 BASIC ROMs. Substitution of line 500 in PERSONAL DIRECTORY will allow it to run with the 3.0 BASIC ROMs.

```
500 POKE 158,3 : POKE 623,13 :
    POKE 624,13 : POKE 625,13
```

Also remove lines 15 and 65 from STUFFIT. 3.0 BASIC doesn't support screen blanking.

WANTED!
Good Articles and Good Photos
MICRO Pays Very Well!

As we increase in size—this issue has 96 pages, 16 more than the last—we can include more articles. However, we are becoming more selective about the articles we accept.

MICRO is committed to covering all of the 6502 systems. To do this well, we need a variety of articles on each system. We can always use more high-quality articles relating to AIM, SYM, KIM, Apple, Atari, PET/CBM, and Ohio Scientific systems. We are especially interested in good articles which apply to 6502 systems in general.

Because we plan to use more illustrations than formerly, we encourage authors to "think pictorially" and to send us good line drawings and black and white photos.

We are also looking for black and white photos which might stand alone, with a brief caption. Photos of 6502 systems in unusual business or professional environments would be especially welcome. Photos used independently of articles will be paid for separately.

For details on how to submit manuscripts for possible publication, ask for *MICRO Writer's Guide*. Write or telephone:

Editorial Department
MICRO
P.O. Box 6502
Chelmsford, MA 01824
617/256-5515

Skyles Electric Works

BASIC Programmer's, Toolkit™, Disk-O-Pro™, Command-O™

For CBM™ Owners Who Want More Fun And Fewer Errors with Their Programs

Here are nineteen commands you'll need, on a single chip you can install in two minutes without tools, **on any CBM or CMB system.** 4KB of ROM firmware on each chip with a collection of machine language programs available from the time you turn on your PET to the time you shut it off.

For CBM 8016 and 8032

BASIC Programmers Command-O™

AUTO^{ed} **DUMP**^{ed} **DELETE**^{ed} **FIND**^{ed} (improved) **HELP**^{ed} **KILL**^{ed} **OFF**^{ed} **TRACE**^{ed} (improved) **RENUMBER**^{ed} (improved) **INITIALIZE**^{BS} **MERGE**^{BS} **MOVE**^{BS} **EXECUTE**^{BS} **SCROLL**^{ed} **OUT**^{ed} **SET**^{ed} **SEND**^{BS} **PRINT USING**^{BS} **BEEP**^{BS}

```
Section A

  100 GOSUB 180
  105 PRINT USING C$, A, B$
  130 INPUT "TIME", D$
  131 INPUT "DAY", E$
  160 IFB<  >C THEN 105
  180 FOR X  1TO9
  183 PRINT Y(X):NEXT
  184 RETURN
  200 I-X/19
READY

RENUMBER 110, 10, 105-184

READY
LIST
  100 GOSUB 150
  110 PRINT USING C$, A, B$
  120 INPUT "TIME", D$
  130 INPUT "DAY", E$
  140 IFB<  >C THEN 110
  150 FOR X=1TO9
  160 PRINT Y(X):NEXT
  170 RETURN
  200 I=X/19
READY
```

```
Section B

MERGE D1 "BUY NOW*"

SEARCHING FOR MEMCMP*
LOADING

READY
RENUMBER 100, 10

READY
FIND B$
110 PRINT USING A$, B$, B$ + C$ + D$
200 B$ = "NOW IS THE TIME"

READY
```

```
Section D

  580 BA-BA   1
  590 RA  123*5X/92+BA*10
  600 IF BA  143 THEN 580
  610 RETURN
  620 C$="PROFIT $#, ###.## DAILY"
  630 PRINT USING C$, PI
  640 D$="LOSS  $#, ###.## DAILY"
  650 PRINT USING D$, LI

RUN

PROFIT $1, 238.61 DAILY
LOSS   $     0.00 DAILY

READY
```

```
Section C

  100 FOR A=4096TO8191:POKEA, B:B  B+1:PRIN
T B:IFB>255 THEN B=B  255:PRINT B

TRACE
```

PRICES:

BASIC Programmers **Toolkit**™ (chip only)	$40.00
BASIC Programmers **Disk-O-Pro**™ (chip only)	$75.00
BASIC Programmers **Command-O**™ (chip only)	$75.00
Interface boards (needed sometimes)	$20.00-$50.00
Instruction Manual (with redeemable $5.00 coupon)	$5.00

Shipping and handling $2.50 USA/Canada, $10.00 Europe/Asia

California residents please add 6% or 6-1/2% sales tax as required

*Reserve your **Disk-O-Pro, Command-O** today*

Toolkit™ immediate delivery, Disk-O-Pro delivery in December, Command-O delivery in January

VISA, MASTERCHARGE ORDERS CALL (800) 538-3083 (except California residents)

CALLIFORNIA ORDERS PLEASE CALL (408) 257-9140

Skyles Electric Works

231 E South Whisman Road
Mountain View, CA 94041
(415) 965-1735

MICRO New Publications

Mike Rowe
New Publications
P.O. Box 6502
Chelmsford, MA 01824

This column lists new publications received for review and also reports on pertinent publication announcements received from book and periodical publishers. Some works mentioned here may be reviewed by MICRO at a later date.

Programming & Interfacing the 6502, With Experiments by Marvin L. DeJong. The Blackburg Continuing Education Series, Howard W. Sams & Co., Inc. (4300 West 62nd St., Indianapolis, Indiana 46168), 414 pages, paperbound.
ISBN: 0-672-21651-5 $13.95

A book for both 6502 microcomputer novices as well as for knowledgeable 6502 users who want to know more about 6502-based systems available today.

CONTENTS: *Part I:* Programming the 6502. *Introduction to Microcomputers*—Objectives; Introduction; What is a Microcomputer?; The 6502 Microprocessor; Introduction to Experiments; Experiments 1 through 3. *Writing and Executing Simple Programs Using Data Transfer Instructions*—Objectives; Introduction; Microcomputer Instructions; Addressing Modes; The Microcomputer Program; A Simple Program; Writing a Program; Loading and Executing a Program; The BRK Instruction; The Single-Step Mode; Introduction to the Experiments; Experiments 1 through 7. *Simple Input/Output Techniques*—Objectives; Introduction; Input/Output Ports; I/O Ports and Data Direction Registers; I/O Port Symbols; Input/Output Programming; JMP Instruction; INC and DEC Instructions; INX, INY, DEX, and DEY Instructions; Introduction to the Experiments; Experiments I through 8. *Logical Operations*—Objectives; Introduction; Logical Operations; AND, ORA, and EOR Instructions; Using ORA, AND, and EOR Instructions to Control Bit Values; Other Uses of Logical Operations; Introduction to the Experiments; Experiments I through 6. *Arithmetic Operations*—Objectives; Introduction; 6502 Processor Status Register; Flag Modification Instructions; ADC Instruction; Multibyte Addition; Decimal Addition; Twos Complement Arithmetic; Signed Number Arithmetic; Signed Arithmetic and Overflow Status Bit; Experiments 1 through 5. *Branches and Loops*—Objectives; Introduction; Branch Instructions; Modifying the Processor Status Register; Branching; Comparison Instructions; Bit Test Instruction; ASCII to Hexadecimal Conversion; Using Branch Instructions for Time Delays; Introduction to the Experiments; Experiments 1 through 6. *Register-Shift Instructions*—Objectives; Introduction; Getting Acquainted With Register-Shift Instructions; A 4-Bit Multiplication Program; An 8-Bit Multiplication Program; Hex to ASCII; Decimal to Hexadecimal; Hexadecimal to Decimal; Experiments I through 8. *Indexed Addressing*—Objectives; Introduction; Absolute Indexed Addressing; Zero-Page Indexed Addressing; Data Tables; Code Conversion Programs; Multiple-Byte Arithmetic; Indirect Addressing; Indirect Indexed Addressing Mode; A Simple Monitor; Indexed Indirect Addressing; Introduction to the Experiments; Experiments I through 7. *Subroutines, the Stack, and Interrupts*—Objectives; Introduction; Subroutines; The Stack; Nested Subroutines; Use of the Stack for Storage; Interrupts; Experiments 1 through 7. *Interval Timers*—Objectives; Introduction; 6530 Interval Timer; 6532 Interval Timer; 6522 Interval Timers; using T2 Timer as a Counter; Using TI Timer; Precision Timing Program; Using TI Timer to Implement Frequency Counter; Making Music Using T1 Timer; Experiments 1 through 8. *Part II:* Interfacing the 6502. *Address Decoding*—Objectives; Introduction; Address Decoding; Address Decoding for R/W Memory; I/O Port Address Decoding; Address Decoding Circuit for 6522 Interface; 6502 Instructions and Device Select Pulses; Introduction to the Experiments; Experiments I through 5. *Control Signals, Ouput Ports, and Applications*—Objectives; Introduction; Clock Signals, Φ_0 (IN), Φ_1 (OUT), and Φ_2(OUT); R/W Control Signal; Using Control Signals for an Output Port; Memory-Mapped, Latched Hexadecimal Display; Memory-Mapped Digital-to-Analog Converter and an Application to Music Synthesis; Other Control Pins on 6502; Experiments I through 5. *Data Bus, Buffering, and Applications*—Objectives; Introduction; Why Buffer?; Memory-Mapped Analog-to-Digital Converter; An ASCII Keyboard Input Port; Experiments I through 5. *Applications*—Introduction; Digital-Analog and Analog-Digital Conversion Using the KIM-1; Employing the KIM-I Microcomputer as a Timer and Data Logging Module; Employing the KIM-I as a Precision Keyer and Automatic Message Sender; Catching Bugs With Lights: A Program Debugging Aid; Lunar Occultation of a Star. *Appendix A: Decimal, Binary, and Hexadecimal Number Systems*—Objectives; Introduction; Numbers; Decimal Numbers; Binary Numbers; Bits, Bytes, and Nibbles; Hexadecimal Numbers; Exercises; Exercise Answers. *Appendix B: Instruction Set Summary. Appendix C: Microcomputer Technical Data. Appendix D: Pin Configuration of Frequently Used SN7400-Series Chips. Appendix E: Pin Configuration of 81LS97. Index.*

Microprocessor Systems Engineering by R.C. Camp, T.A. Smay, and C.J. Triska. Matrix Publishers, Inc. (30 NW 23rd Place, Portland, OR 97210), 1979, viii, 642 pages, hardbound.
ISBN: 0-916460-26-6 $29.95

An introduction to microprocessors for students of electrical engineering. Focuses on the AIM-65 microcomputer, based on the 6502 microprocessor.

CONTENTS: *Introduction to Microcomputer-based Design*—Evolution of the Microcomputer; Microprocessor Applications; Engineering Design of Microcomputer-Based Products; Educational Demands Created by the Microprocessor; Objectives of this Book. *General Aspects of Microprocessor-Based Systems*—Microprocessors and Microcomputers; Classification of Computers and Computer Systems; General Features of Microcomputer-Based Systems; Information Flow in Microcomputers; Central Processor Hardware Elements; Addressing Modes; Microprocessor Instruction Sets; Microprocessor Word Length; Symbolism in Digital Computers; Arithmetic Operations in Microcomputers; Interrupts and Subroutines; Technological Factors in Microprocessors. *The 6502 Microprocessor and Peripheral Parts*—Introduction to 6502; Programming Model; Data Paths; Concept of Operation of 6502 Instructions; Complete Description of Operation Codes; 6502 Specifications; Peripheral Interface Chips; Example Problems. *Software Aids*—Introduction; The Software Design Process; Elements of Program Translation; Text Editors; Simulators; Special Program Debug Features; In-circuit Emulation; Logic State Analyzers; Prom Programmers. *Microcomputer Interfacing and System Design*—Introduction; Guidelines for System Design; Miscellaneous Advice on System Design; Interfacing Examples; Input; Output-TTL, Speed, Bits, Serial Parallel Conversions; Address Maps and Organization; Memory and I/O Selection; System Design Examples. *Introduction to the 6502 Microprocessor*—Introduction; Principal Characteristics; Some 6502 and 6800 Differences; 6800 Programming; Electrical Characteristics of the 6800; 6800 Microcomputer Example; Example Problems. *Introduction to 8080 Microprocessor*—Characteristics; 8080 Architecture and Programming Model; Data Paths; 8080 Instruction Set; 8080 Example Program; Electrical Characteristics of the 8080. *A Case Study—The AIM 65*—Introduction; Memory Interfacing; LED Display Interface; Keyboard Interfacing; Printer Interface; Teletype Interface; Interrupt Handling; User Port Interfacing; Monitor Subroutines. *A: A 6502 Based Microprocessor—The AIM-65; B: The System 65—A 6502 Development System; C: 2's Complement Arithemetic in the 6502; Index.*

MICRO

MICRO

Microprocessors in Medicine: The 6502

By Jerry W. Froelich, M.D.

The column this month and last month, written together with Jack W. Smith, M.D., informs readers on various uses of computers in medical education and provides examples of how the 6502 microprocessor is able to perform tasks in medical education nearly as well as large computer systems. (Dr. Smith is a Clinical Fellow in Pathology, Instructor in Allied Health, and Ph.D candidate in Computer Science at Ohio State University, Columbus, Ohio.)

Last month, we described various types of programs: computer-aided instruction (CAI), computer-assisted evaluation (CAE), and simulation. This month, we will cover "APPLE-ED," a program produced by Computer Methods Supplements (CMS, of Ann Arbor, Michigan) for educating physicians and technologists in nuclear cardiology.

Last month, no mention was made of problems with computers in medical education, and there are some disadvantages. Therefore, before we move on to the APPLE-ED, let's discuss the problems. The lack of uniformity in computer systems makes transfer of programs difficult. Lack of agreement of information content in programs increases the difficulty of transfer. At present, there is no formal production-distribution system to make available the programs which exist. Prior to the development of microprocessors, computer-aided instruction required large computer systems. Currently, there are three large medical institutions with large investments in computers for use in medical education. They are the Massachusetts General Hospital in Boston, Massachusetts, the University of Illinois at Urbana, and Ohio State University in Columbus, Ohio.

At all three institutions, the students interact with a CRT terminal which is connected to the host computer via telephone lines. Personnel not based at the institutions have limited access.

With the advent of microprocessors and languages such as BASIC and PILOT, the production-distribution system for medical knowledge is potentially enhanced. However, consistently-updated, high-quality material is still not available.

One application which I would like to describe is called, APPLE-ED. This is a microprocessor-based system produced by Computer Methods Supplements. APPLE-ED is, as the name implies, written for an APPLE computer.

The programs are written in BASIC and are designed so that once the system has been "booted up," the program will sequence itself through the three floppy disks to a fourth disk which is a testing program. The APPLE-ED programs present a series of complete lectures on the use of computers in Nuclear Medicine and Nuclear Cardiology. (Perhaps at a future time we can give an in-depth description of computer applications in nuclear medicine.) The lecture content is based on the textbook entitled, *Computer Methods: The Fundamentals of Digital Nuclear Medicine,* published by C.V. Mosby Company and edited by David E. Liberman of CMS.

The three sections of the APPLE-ED programs are: "Basic Elements of Computers"; "Display and Processing of Nuclear Medicine Studied"; and "Nuclear Cardiology Techniques." The examination disk is structured so that the participant takes an examination. The results are recorded on the disk, and at some point the disk is returned to CMS for grading.

At Massachusetts General Hospital, the residents and visiting fellows use the system and have found it an excellent introduction to the field of computers in nuclear medicine.

Summary

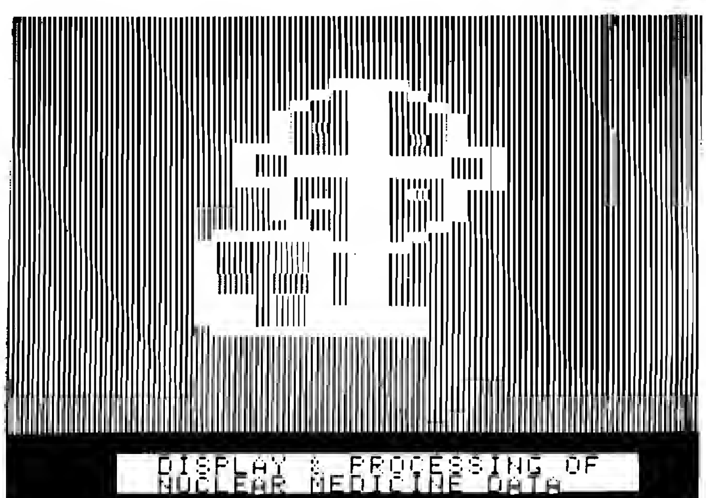
We have reviewed the types of computer programs in medical education (CAI, CAE, etc.) and concluded with a description of APPLE-ED, a 6502 application. APPLE-ED requires no hardware more sophisticated than a microcomputer and a disk drive. The disk is only required to make the loading of the programs quicker. If you are willing to tolerate the speed of tape, then tape is a viable storage medium.

The programs are written in BASIC, which allows text to be printed and which queries the user on the content of the material presented. Based upon the participant's response, the program then branches. It either progresses further into the material or reinforces points, and reviews material that has already been covered.

To quote a user of APPLE-ED, "It sounds quite beneficial for us. We don't have a training program for technologists, but our residents could benefit from the continuing education, and it's a fraction of the cost of sending someone to a clinical seminar. Thanks for your help!" (APPLE-ED can be either purchased or leased from Computer Methods Supplements.)

Note to Readers

Thank you for the response to my first column. I will begin to incorporate suggestions into future columns. Anyone wishing to share his application, please drop me a line. I must apologize, but because of the number of letters, if you would like a response please include a stamped, self-addressed envelope. Please send all correspondence to me at: c/o Massachusetts General Hosptial, Boston, MA 02114.



Apple* Software

From Instant Software

Santa Paravia and Fiumaccio

Buon giorno, signore!

Welcome to the province of Santa Paravia. As your steward, I hope you will enjoy your reign here. I feel sure that you will find it, shall we say, profitable.

Perhaps I should acquaint you with our little domain. It is not a wealthy area, signore, but riches and glory are possible for one who is aware of political realities. These realities include your serfs. They constantly request more food from your grain reserves, grain that could be sold instead for gold florins. And should your justice become a trifle harsh, they will flee to other lands.

Yet another concern is the weather. If it is good, so is the harvest. But the rats may eat much of our surplus and we have had years of drought when famine threatened our population.

Certainly, the administration of a growing city-state will require tax revenues. And where better to gather such funds than the local marketplaces and mills? You may find it necessary to increase custom duties or tax the incomes of the merchants and nobles. Whatever you do, there will be far-reaching consequences...and, perhaps, an elevation of your noble title.

Your standing will surely be enhanced by building a new palace or a magnificent *cattedrale*. You will do well to increase your landholdings, if you also equip a few units of soldiers. There is, alas, no small need for soldiery here, for the unscrupulous Baron Peppone may invade you at any time.

To measure your progress, the official cartographer will draw you a *mappa*. From it, you can see how much land you hold. how much of it is under the plow and how adequate your defenses are. We are unique in that here, the map IS the territory.

I trust that I have been of help, signore. I look forward to the day when I may address you as His Royal Highness, King of Santa Paravia. *Buona fortuna* or, as you say, "Good luck". For the Apple 48K.

Order No. 0174A $9.95 (cassette version).
Order No. 0229AD $19.95 (disk version).

TO ORDER

SEE YOUR LOCAL INSTANT SOFTWARE DEALER OR USE THE ORDER FORM BELOW

For Fast Service *call now* Toll-Free
1-800-258-5473

Apple Cassettes

0018A Golf.............................. $7.95
0025A Mimic.............................. $7.95
0040A Bowling/Trilogy.................... $7.95
0073A Math Tutor I....................... $7.95
0079A Oil Tycoon......................... $9.95
0080A Sahara Warriors.................... $7.95
0088A Accounting Assistant............... $7.95
0094A Mortgage w/Prepayment Option/ Financier.......................... $7.95
0096A Space Wars......................... $7.95
0098A Math Tutor II...................... $7.95
0174A Santa Paravia and Fiumaccio......... $9.95
0148A Air Flight Simulation.............. $9.95

We Guarantee It!

Instant Software Guarantee

OUR PROGRAMS ARE GUARANTEED TO BE QUALITY PRODUCTS. IF NOT COMPLETELY SATISFIED YOU MAY RETURN THE PROGRAM WITHIN 60 DAYS. A CREDIT OR REPLACEMENT WILL BE WILLINGLY GIVEN FOR ANY REASON.

109

Name ______

Address ______

City ______ State ______ Zip ______

☐ Check ☐ Money Order ☐ VISA ☐ AMEX ☐ Master Charge

Card No. ______ Exp. Date ______

Signed ______ Date ______

Order your Instant Software today!

Quantity	Order No.	Program name	Unit cost	Total cost
		Shipping and handling		$1.00
			Total order	

Instant Software Inc.
Peterborough, N.H. 03458

ATARI BITS

128 colors on the screen at one time? Explore some of the many, extraordinary features ATARI provides, but doesn't always tell you about.

Len Lindsay
1929 Northport #6
Madison, WI 53704

Atari Safari

The ATARI 400 and 800 computers are quite amazing machines. The more I learn about them, the more amazing they seem. They offer 3 different text modes and 6 different graphic modes. But they don't tell you that you can mix and match different modes on the screen at the same time. BASIC allows you to use any 6 of the 128 possible colors at one time, but did you know that it is possible to have all 128 on the screen at one time?

I hope to share information and programming tips with you, to help you use your ATARI computer for amazing things. So let's get started. This month I will explain some quick little programming tricks and methods. Watch for a future article explaining the things I have mentioned. If you have any ATARI programming tips you would like to share please contact me:

Len Lindsay
c/o MICRO, P.O. Box 6502
Chelmsford, MA 01824

TV Screen Protect Feature

If you leave the same exact picture on your TV screen for an extensive period of time, it will "burn" an image into the screen. This can ruin your good color TV set. ATARI realized this when they designed their computers, and built in a feature to protect your screen. If you don't hit a key on the keyboard for several minutes, the computer goes into attract mode, changing the color registers every several seconds. The screen image remains the same, just the colors change, dark to light, to medium, etc. Changing the colors helps to prevent the image from "burning" into your screen.

Memory location 77 is used by the ATARI as a sort of counter, and is reset to 0 each time the keyboard is accessed. Every few seconds it increments by one. When it reaches 128, the computer goes into attract mode. You may write a program that uses a joystick for input, or one that continually generates dynamic computer art. Thus several minutes will go by without any keyboard access, but you do not want the colors to begin changing. To prevent it, just add this one line in the appropriate place:

```
200 POKE 77,0 : REM RESET
    ATTRACT MODE COUNTER
    TO ZERO
```

When this line is executed, the computer thinks a key has been hit, and the counter is reset back to zero. Of course you can do the exact opposite and start attract mode at any time as well. Just use this line:

```
500 POKE 77,128 : REM START
    ATTRACT MODE
```

When executed, the computer will go into attract mode, changing the color registers every few seconds, providing you with an easy method to have your screen change colors.

Keyboard Buffer

The ATARI is always looking at its keyboard, even while doing other things under program control. It remembers the last key hit. For example, if you hit the "A" key while the computer is drawing something on the screen, it will remember that you hit the key. ENTER THIS PROGRAM:

```
10 REM BUSY WORK
   PROGRAM
20 FOR DELAY=1 TO 999
30 NEXT DELAY
```

Now RUN the program, and hit the "A" key while it is thinking. When the program is done, watch an A appear on the screen. The computer remembered that you hit the key. Try it again, but hit several keys. Only the last key you hit will be remembered. This feature has its uses, but can be annoying at times. For instance, an INPUT statement in your program will use the key in the keyboard buffer as the first character of the INPUT. Try the following program:

```
10 REM CHEATING ON THIS IS
   EASY
15 PRINT "HIT A KEY WHEN I
   SAY GO"
20 FOR DELAY = 1 TO
   RND(1)*500+500
30 NEXT DELAY
40 PRINT
50 PRINT "GO"
60 OPEN #1,4,0,"K:" :REM
   OPEN THE KEYBOARD FOR
   A GET COMMAND
70 GET #1,KEY :REM WAIT TILL
   A KEY IS HIT
80 CLOSE #1 :REM CLOSE FILE
90 PRINT "THANK YOU"
```

RUN the program. Hit a key after it says GO, and it will respond THANK YOU. However, you can cheat very easily. Just hit a key immediately after you RUN the program.

There is an easy way to prevent cheating of this nature. Add this one line to your program and try to cheat:

```
65 POKE 764,255 : REM CLEAR
   KEYBOARD BUFFER
```

It is always a good idea to use this statement just before any INPUT in your program, for example:

```
100 PRINT "WHAT IS YOUR
    AGE":
110 POKE 764,255
120 INPUT AGE
```

Clearing the buffer makes sure that you have a fresh start for your INPUT.

Dynamic Keyboard

You just saw how to clear the keyboard buffer. Now, what if you would like to add something to it? This may seem like a silly idea, but I will show that it is very practical. Your program can list an individual line of your program to disk with this command:

```
600 LIST "D:LIST",10 :REM
    LIST LINE 10 TO DISK—
    FILE NAME 'LIST'
```

Your program will continue executing after this command is executed. Line 10 can be a line of DATA or some other varying aspect of your program.

You also can ENTER new program lines from disk, *while the program is running*. The program can thus change itself as it RUNs. This line will ENTER a new line 10 into the program (if a line 10 was previously put to disk with the LIST command):

```
700 ENTER "D:LIST" : REM
    ENTER NEW PROGRAM
    LINES FROM DISK
```

However, after the program executes an ENTER command, it stops program execution, and waits for keyboard commands. To have the program continue executing you would then have to type in:

```
CONT
```

There is a way to have all this done automatically for you. All we have to do is trick the computer into thinking that you just typed in CONT (or a GOTO command) and hit RETURN. First, here is a program that will record line 99 in your program onto disk:

```
10 REM LIST LINE 99 TO DISK
20 PRINT "RECORDING LINE
   99 TO DISK"
30 LIST "D:LINE99",99
40 PRINT "DONE —"
99 PRINT "THIS IS LINE 99"
```

RUN this program. It will save line 99 to DISKETTE. We will use it for input to the next program.

Now here is a program that will ENTER program lines from diskette, thus altering itself while it is running. (Remember to type NEW first.)

```
10 REM ENTER PROGRAM
   LINE FROM DISKETTE
20 PRINT "READING LINE
   FROM DISK"
30 PRINT "[DOWN][DOWN]
   GOTO 50[UP][UP]";
35 POKE 764,12 : REM TRICK
   ATARI TO THINK YOU HIT
   THE RETURN
40 ENTER "D:LINE99"
50 PRINT "DONE —"
```

Run this program with the diskette still in your drive. Surprise! It printed DONE and also THIS IS LINE 99. Try the program without line 35. It still will add line 99 from diskette, but quits executing at that point.

NOTE: the above program requires cursor movements to be programmed into line 30. [DOWN] means to program in one cursor down. Do not type it in literally. To program in a cursor down, simply hit the ESC key, then hit CONTROL CURSOR DOWN. The same applies to [UP], which means program in one cursor up.

ATARI High Resolution Text

The ATARI lets you enjoy every high resolution graphics mode—GRAPHICS 8. Normally, the screen can only plot points. However, this routine can be used to put text on the screen along with your graphic display. Here is the subroutine:

```
23000 ZL = PEEK(560) +
      PEEK(561)*256
23020 ZM = PEEK(ZL+4)
      +PEEK(ZL+5)*256
23030 FOR ZW = 1 TO
      LEN(ZA$)
23040 ZT = 57344 +
      ((ASC(ZA$(ZW,
      ZW)-32)*8)
23050 ZC = ZM + ZY*40+
      ZX+(ZW-1)
23060 FOR ZR=0 TO 7
23070 POKE ZC+ZR*40,
      PEEK(ZT+ZR)
23080 NEXT ZR
23085 ZY=ZY+ZZ
23090 NEXT ZW
23099 RETURN
```

Notice that all variables used begin with Z. That means that conflicts with variables in your main program will probably not occur, or you will be alerted to each Z variable you use.

ZA$ is the test to be printed
ZX is the column to start printing (0 – 39)
ZY is the row to start printing (0 – 191)
ZZ is the slant of the printed line
ZL, ZT, ZC, ZW, ZR are temporary variables used

To utilize the routine, simply set ZA$ to the text you want, set ZX and ZY to the starting coordinate and GOSUB 23000. If you want your text printed in a straight line, ZZ should equal 0. Try ZZ = 1 and ZZ = – 1. An example line to call the subroutine might be this:

```
500 ZA$="TESTING TEXT"
    :ZX=5:ZY=40:ZZ=1
    :GOSUB 23000
```

You might wish to fool with the subroutine to make it print wavy lines:

```
23085 ZY=ZY+ZZ:ZZ=-ZZ
```

You can also fool with the concepts in the routine to print the text in vertical lines as well as horizontal.

Upper, or Lower, or Graphics

Graphics mode 2 is the large TEXT mode. ALL PRINT#6 commands are printed to the screen in extra large size. For example:

```
10 GRAPHICS 2 : REM LARGE
   TEXT MODE
20 PRINT#6;"TESTING"
```

The word TESTING is printed on the screen in large gold letters. Now try the same program, but change line 20 so that the word you print is in lower case letters. Example:

```
10 GRAPHICS 2 : REM LARGE
   TEXT MODE
20 PRINT#6; "testing"
```

This time the word TESTING is printed on the screen in large light green letters. But the letters are all upper case. Next, try typing the word TESTING in reverse field. (Hit the ATARI symbol key just before typing the first letter in the word TESTING. After typing the word TESTING in reverse field, hit the ATARI symbol key again to be back to normal.) Try printing both UPPER and lower case letters in reverse field.

You now see that you can get 4 different colored UPPER case large letters on the screen by varying how you type them in the PRINT statement. But

what if you really want lower case letters on the screen? Simple, one POKE will do the trick:

```
POKE 756,226 : REM LOWER
      CASE CHARACTER BASE
```

Type in that POKE statement after running your program. Surprise!

Everything Is Upside Down

With one simple command you can turn all of the characters on the screen upside down. And from that point on, all characters printed will also be upside down. Just enter this command:

```
POKE 755,4 : REM UPSIDE
      DOWN CHARACTERS
```

Now type anything you want and watch it come out upside down. Hit the SYSTEM RESET button, and all will return to normal. Or enter this command:

```
POKE 755,2 : REM BACK TO
      NORMAL
```

This can be used within a program for special effects. Try this quick example:

```
 10 GRAPHICS 2 : REM LARGE
    TEXT MODE
 20 POSITION 9,9 : REM
    STARTING POINT FOR
    PRINTING WORD
 30 PRINT#6;"TESTING"
 40 X=2 : REM INITIALIZE
    VARIABLE FOR POKES
100 POKE 755,X : REM UPSIDE
    DOWN AND THEN BACK
    AGAIN
110 FOR DELAY=1 TO 200 :
    REM PAUSE A SECOND
120 NEXT DELAY
130 X=6-X : REM SWITCH
    FROM 2 TO 4 AND THEN
    BACK FROM 4 TO 2
140 GOTO 100 : REM DO IT
    AGAIN
```

I think you will enjoy the example that was just mentioned. Hit the BREAK key while the word is upside down. Isn't it interesting that the message

```
STOPPED AT LINE 110
```

is often printed upside down?

MICRO

MICRO Letterbox

Dear Editor:

In the April MICRO, (23:65) Mr. Earl Morris wrote an interesting article on OSI BASIC in ROM. I have a disk system, but I was interested in the same for my Disk BASIC, which, I might add, really lacks a memory map. So I started to use the program under the assumption the two BASICs were generally the same, and they are—right down to the AND, OR, GREATER, LESS, and EQUAL. The only difference being the area of memory occupied. The following is what is required to make Earl's program work for the Disk BASIC.

```
 10 Q=1:D=644
 20 FOR C=512 TO 612 STEP 2
115 D=776
120 FOR C=614 TO 641
      STEP 3
```

Note: By changing Line 120 in the ROM version to:

```
FOR C=41062 TO 41089
      STEP 3
```

you will find the missing AND, OR, GREATER, LESS, and EQUAL.

Les Cain
2606 Grand Ave.
Grand Junction, Colorado 81501

Dear Editor:

I appreciated Mr. Childress' article in the May issue (24:45) on entering lowercase and punctuation into Applesoft strings, but I would like to point out that it *is* possible to enter punctuation into INPUT strings without the use of special routines. If the string is enclosed in quotation marks (e.g., "HALL, C.W.") all characters between the quotation marks will be accepted as part of the string. The only character I have not been able to enter this way is the quotation mark ("), which delimits the string. However, if the string is not delimited with quotation marks, they may be entered anywhere in the string.

I discovered this use of the quotation marks in Applesoft when I needed to enter a list of names, last name first, and remembered how such strings were entered in BASIC on the DEC and CDC machines I programmed in college. I know this information will be a great help to all those Applesoft users who have been trying to enter commas and colons into their strings.

Charles W. Hall
3262 Olive Place
Fort Worth, Texas 76116

Dear Editor:

I thought your readers might be interested in a slightly more enhanced version of EDITPLUS. All of the original features are the same, plus the following: (1) 'ESC' 'H' will clear and home the screen, (2) 'ESC' 'P' performs a POKE 33,33 to change the screen width to 33 columns for easier editing of literals (string values inside quote marks) and (3) 'ESC' 'N' returns the screen width to a normal 40 columns. Listed below is a memory dump of the improved version, located at $300. Just type it in and 'CALL 768'. It's set up for use with 48K of memory and 3.2 DOS. To revise it for other configurations, see the EDITPLUS article in the June issue of MICRO and the necessary revisions will be apparent.

It sure makes editing easier, and it works in Integer BASIC, Applesoft and in the Monitor. Also, the enhancements make it of value to the owners of APPLE II Plus computers.

```
0300— A9 13 85 38 A9 03 85 39
0308— A9 69 85 36 A9 03 85 37
0310— 4C 51 A8 20 1B FD C9 9B
0318— F0 0B 60 38 E9 C9 A8 B9
0320— 8C 03 20 2C FC A4 24 B1
0328— 28 48 29 3F 09 40 91 28
0330— 68 20 1B FD C9 C8 D0 02
0338— A9 C0 C9 D0 F0 18 C9 CE
0340— F0 10 B0 19 C9 C9 90 15
0348— C9 CC D0 CF 20 51 A8 4C
0350— 65 FF A9 28 DO 05 20 42
0358— FC A9 21 85 21 38 20 2C
0360— FC A4 24 8C 5B AA 4C 0C
0368— FD 84 35 C9 8D D0 18 AC
0370— 00 C0 10 13 C0 93 D0 0F
0378— 2C 10 C0 AC 00 C0 10 FB
0380— C0 83 F0 03 2C 10 C0 A4
0388— 35 4C F0 FD C4 C2 C1 FF
0390— C3
```

Craig Peterson
1743 Centinela Ave. #102
Santa Monica, California 90404

Continued on page 94

OSI **SOFTWARE FOR OSI** OSI

A JOURNAL FOR OSI USERS!!

The Aardvark Journal is a bimonthly tutorial for OSI users. It features programs customized for OSI and has run articles like these:

1) Using String Variables.
2) High Speed Basic On An OSI.
3) Hooking a Cheap Printer To An OSI.
4) An OSI Disk Primer.
5) A Word Processor For Disk Or Tape Machines.
6) Moving The Disk Directory Off Track 12.

Four back issues already available!
$9.00 per year (6 issues)

ADVENTURES

Adventures are interactive fantasies where you give the computer plain English commands (i.e. take the sword, look at the control panel.) as you explore alien cities, space ships, ancient pyramids and sunken subs. Average playing time is 30 to 40 hours in several sessions. There is literally nothing else like them — except being there yourself. We have six adventures available.

ESCAPE FROM MARS — Explore an ancient Martian city while you prepare for your escape.

NUCLEAR SUBMARINE — Fast moving excitement at the bottom of the sea.

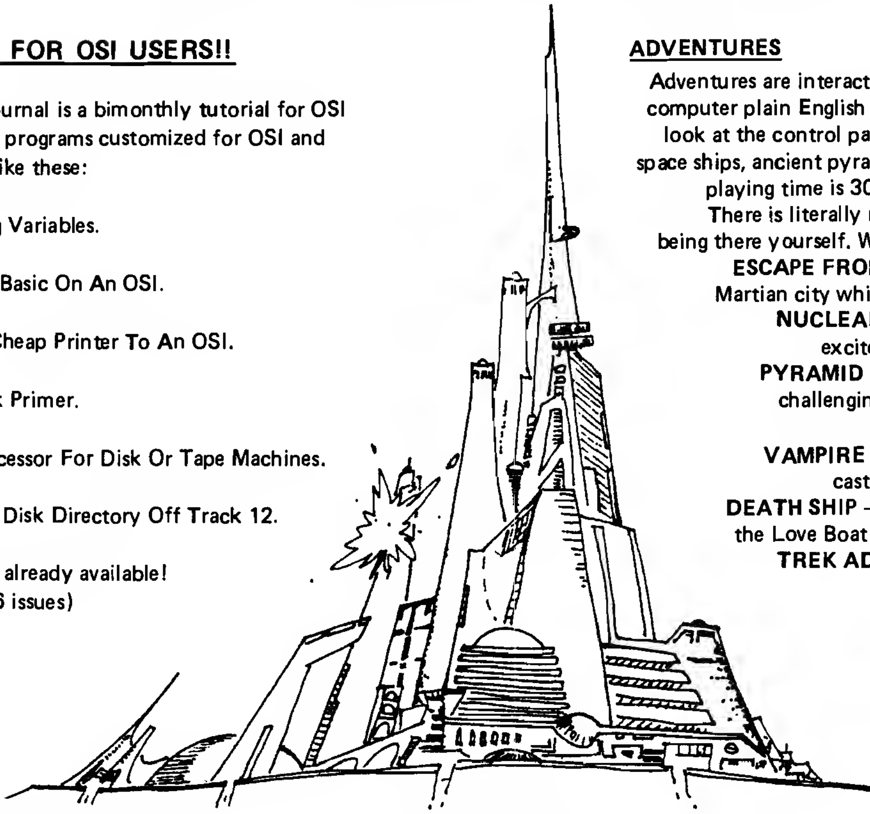
PYRAMID — Our most advanced and most challenging adventure. Takes place in our own special ancient pyramid.

VAMPIRE CASTLE — A day in old Drac's castle. But it's getting dark outside.

DEATH SHIP — It's a cruise ship — but it ain't the Love Boat and survival is far from certain.

TREK ADVENTURE — Takes place on a familiar starship. Almost as good as being there.

$14.95 each



NEW SUPPORT ROMS FOR BASIC IN ROM MACHINES

C1S — for the C1P only, this ROM adds full screen edit functions (insert, delete, change characters in a basic line.), Software selecteble scroll windows, two instant screen clears (scroll window only end full screen.), software choice of OSI or standerd keyboard format, Bell support, 600 Baud cassette support, end a few other features. It plugs in in place of the OSI ROM. NOTE: this ROM also supports video conversions for 24, 32, 48, or 64 characters per line. All that end it sells for a mesly $39.95.

C1E/C2E for C1/C2/C4/C8 Basic in ROM machines.
This ROM adds full screen editing, software selectable scroll windows, keyboerd correction (software selectable), and contains both an extended machine code monitor and a fix for the string hendling bug in OSI Basic!! It has breakpoint utilities, machine code load and save, block memory move end hex dump utilities. A must for the machine code programmer replaces OSI support ROM. Specify system! $59.95

STRING BUG FIX (replaces besic ROM chip number 3)
All this chip does is to replace the third basic ROM and correct the errors that were put into the ROM mask. $19.95

DATA SHEETS

OS65D LISTING
Commented with source code, 83 pages. $24.95

THE (REAL) FIRST BOOK OF OSI
65 packed pages on how OSI basic works. Our best selling data sheet. $15.95

OSI BASIC IN ROM
Ed Carlson's book of how to program in besic. Now available from Aardvark. $8.95

P.C. BOARDS

MEMORY BOARDS!! — for the C1P. — and they contain perellel ports!
Aardvarks new memory boerd supports 8K of 2114's and hes provision for a PIA to give e parallel ports! It sells es a bare board for $29.95. When essembled, the board plugs into the expension connector on the 600 board. Available now!

REAL SOUND FOR THE C1P — and it's cheap!
This bare board uses the TI sound chip to give real arcade type sound. The board goes together in e couple of hours with about $20.00 in perts. Bare board, plans, and sample program — $15.95

ARCADE AND VIDEO GAMES

ALIEN INVADERS with machine code moves — for fast ection. This is our best invaders yet. The disk version is so fast that we had to add selectable speeds to make it playeble.
Tape - $10.95 — Disk - $12.95

TIME TREK (8K) — real time Startrek action. See your torpedoes move across the screen! Real grephics — no more scrolling displays. $9.95

STARFIGHTER — a real time space war where you face cruisers, bettleships and fighters using a variety of weapons. Your screen contains working instrumentation and a real time display of the alien ships. $6.95 in black and white - $7.95 in color and sound.

SEAWOLFE — this one looks like it just stepped out of the arcades. It feetures multiple torpedoes, severel target ships, floating mines and real time time-to-go and score displays. — $6.95 in black and white $7.95 in color end sound.

SCREEN EDITORS

These programs all ellow the editing of basic lines. All assume thet you ere using the standard OSI video displey end polled keyboerd.

C1P CURSOR CONTROL — A program that uses no RAM normally available to the system. (We hid it in unused space on page 2). It provides real backspace, insert, delete and replace functions and an optional instant screen clear.
$11.95

C2/4 CURSOR. This one uses 366 BYTES of RAM to provide e full screen editor. Edit and change lines on any part of the screen. (Basic in ROM systems only.)

FOR DISK SYSTEMS — (65D, polled keyboard and standerd video only.)

SUPERDISK. Contains a basic text editor with functions similar to the above programs and also contains a renumberer, variable table maker, search and new BEXEC* programs. The BEXEC* provides a directory, creete, delete, and change utilities on one track and is worth having by itself. — $24.95 on 5" disk - $26.95 on 8".

DISK UTILITIES

SUPER COPY — Single Disk Copier
This copy program makes multiple copies, copies track zero, end copies all the tracks that your memory can hold at one time — up to 12 tracks at e pess. It's almost as fast as dual disk copying. — $15.95

DISK CATALOGER
This utility reads the directory of your disks end makes up an elphabetic list off all your programs end what disks they are on. $14.95

MACHINE CODE RENUMBERER
(C2/4-MF only)
Renumbers all or part of a program et machine code speeds. — $15.95

VISA

This is only a partial listing of what we have to offer. We now offer over 100 programs, data sheets, ROMS, and boards for OSI systems. Our $1.00 catalog lists it all and contains free program listings and programming hints to boot.

Aardvark Technical Services • 1690 Bolton • Walled Lake, MI 48088
(313) 669-3110 or (313) 624-6316

Relocating OSI ROM BASIC Programs

This BASIC program relocator will help users of Ohio Scientific computers with BASIC in ROM to better understand how their Microsoft BASIC and monitor are used.

William L. Taylor
246 Flora Road
Leavittsburg, OH 44430

Two articles recently published in MICRO inspired this article. The article entitled "Some Useful Memory Locations and Subroutines for OSI BASIC in ROM" by S.R. Murphy, that appeared in the November 1979 MICRO (18:9), gave a list of zero page locations used by the Ohio Scientific Challenger computers as a scratch pad memory. This memory map along with the article "Relocating PET BASIC Programs" by Michael Tulloch, that appeared in the December 1979 MICRO (19:25), inspired me to try a BASIC program relocater for Ohio Scientific computers.

To begin with, since Microsoft wrote the BASIC that is used in Ohio Scientific Challengers and Commodore PET computers, it would seem there would be similarities. This is true. Both versions of BASIC use low memory in the same manner as a scratch pad. Zero page, for example, is used as a scratch pad to store BASIC's parameters. A list or memory map for the Challengers and PET is listed in table 1. From the table it can be seen that both the Challengers and PET use the same pointers. There are differences between the version for the PET and the one for the Challengers and how they use some locations in zero page; but both versions use identical pointers for memory allocation, for the beginning of BASIC work space, etc. One difference between the versions is that Ohio Scientific uses page 3 of the system memory as a part of BASIC program memory workspace.

Ohio Scientific computers with BASIC in ROM perform the same tests on memory as do PETs. That is, hex 24 is loaded into memory locations from 0301 hex upwards, depending on the memory size. When Ohio Scientific's BASIC in ROM machines are brought up under cold start, the user may define memory size or allow BASIC to utilize all the available memory in the system from hex 0301 upward.

After BASIC tests memory for available space and determines the upward limit, this available size is stored in a zero page location called the memory size pointer. On initialization, there are several other parameters set up in the scratch pad memory in zero page under ROM BASIC. These parameters are called pointers. We have already used this term and have defined two of these pointers. Ohio Scientific ROM BASIC always sets its pointers to begin at 0301 hex or 769 decimal for a starting point.

Table 1:
Relocating Ohio Scientific BASIC Programs

Similarities in PET and Ohio Scientific Scratch Pad

The Commodore PET has BASIC program work space set to begin at 0401 hex. Ohio Scientific has the BASIC work space set to begin at 0301 hex.

Scratch Pad Area

	PET		OSI	
BASIC START	122 dec.	7A hex	121 dec.	79 hex
	123 "	7B "	122 "	7A "
Single	124 "	7C "	123 "	7B "
Variable	125 "	7D "	124 "	7C "
Array	126 "	7E "	125 "	7D "
Variable	127 "	7F "	126 "	7E "
Array Space	128 "	80 "	127 "	7F "
Available	129 "	81 "	128 "	80 "
String	130 "	82 "	129 "	81 "
Bottom	131 "	83 "	130 "	82 "
String	132 "	84 "	131 "	83 "
Top	133 "	85 "	132 "	84 "
Memory	134 "	86 "	133 "	85 "
Size	135 "	87 "	134 "	86 "
Present	136 "	88 "	135 "	87 "
BASIC Line	137 "	89 "	136 "	88 "
Line #	138 "	8A "	137 "	89 "
at BREAK	139 "	8B "	138 "	8A "
Pointer For	140 "	8C "	139 "	8B "
CONT.	141 "	8D "	140 "	8C "

As was noted by Mr. Tulloch in his article on relocating PET BASIC programs, there are several pointers in the scratch pad memory that must be changed to initiate a relocation of BASIC programs. These pointers are: the beginning of BASIC program; the beginning of the single variable; the beginning of array variables; the available space for DIM array variable; and, finally, the top of strings and the bottom of strings. All of these pointers must be changed to point to the location for a BASIC program, if a new starting area is to be used. As stated before, the listing in table 1 will show the location in the scratch pad where the pointers are located. In addition, I will describe how to use these pointers to allow you to relocate your Ohio Scientific BASIC programs.

The Ohio Scientifc Microsoft BASIC in ROM uses addreses hex 79 and 7A or decimal 121 and 122 as the BASIC start pointer locations. On a BASIC cold start, these locations contain a pointer that points to hex 0301 or decimal 769. The data stored in these locations must be in the 6502 format, that is, low byte followed by the high byte (for example, 0079 01 007A 03). All the pointer locations are two bytes wide and must have their data in this format. As an example, if you wished to have your BASIC program start at, say, hex 0400, then this address would have to be stored in 0079 and 007A as 00,04. To relocate your programs to start at 0400 hex, you would have to change all the pointers in the same manner. The seven pointers that must be changed are listed in table 1.

As an example, let's reinitialize the pointers in zero page for a BASIC program start address to begin at 0800 hex. To have the program begin at 0800 hex, we will need to change the high byte of the pointers for BASIC program start, simple variable start, array variable start, available space, and string top and string bottom. To make this change, bring up BASIC in cold start. Reset the computer. Bring up Monitor Mode by typing "M" on the keyboard. Once in Monitor Mode, you can call up the pointer addresses and change the data, to point to the new BASIC program starting point. In address mode, call up 007A hex. Enter Data Mode by typing a slash (/) on the keyboard. Now load the required data at this address, in this case hex 08. Enter hex 08 at locations 007C, 007E, and 0080. Return to Address Mode. Call up 0800 hex. Examine the data stored at 0800. If this data is not 00, then change this data to read hex 00. Reset the computer. Call up BASIC in warm start with "W" on the keyboard. Now type NEW followed by RETURN. If all went well the computer should respond with OK. Your BASIC work space has now been changed to begin at page 8 and your BASIC programs will be written upward from this point.

The last example is only one method of re-initializing the pointers. A different approach to this task is demonstrated in program listing 1. This program provides a BASIC and machine language program that can be saved on cassette tape and can be loaded into the C1P or other Ohio Scientific system when the need arises. Refer to listing 1 for the following description.

The BASIC portion of the program is used as an executive in connection with the machine language routine that actually does the work in initializing the scratch pad area pointers. The machine code program is stored in the memory area between 0200 hex and 0300 hex. This area in memory is little used and rarely mentioned in most articles. The memory area between 0222 and 02FF hex is not used by BASIC or the Ohio Scientific monitor and is free for machine language routines or any other machine coded programs that can fit into this area. This is a perfect location for our machine code routine used in this program. Once the machine code routine is stored in this area, it can be called at any time the need

BASIC Program Relocator

```
5 REM OSI ROM BASIC PROGRAM RELOCATOR
7 PRINT" ROM BASIC PROGRAM RELOCATOR"
10 FOR Q=546 TO 573
20 READ P: POKE Q,P
30 NEXT Q
50 INPUT" START";A
60 POKE 547,A
70 POKE 570,A
80 POKE 11,34:POKE 12,2
90 X=USR(X)
100 DATA 169,0,133,122,133,124,133,126
110 DATA 133,128,133,144,133,173,133,165
120 DATA 133,167,133,196,169,0,141,0,0
130 DATA 76,0,0
```

Disassembled Object Code
Located at 0222 through 023D.

```
0222 A9 00        LDA   #$00
0224 85 7A        STA   $007A
0226 85 7C        STA   $007C
0228 85 7E        STA   $007E
022A 85 80        STA   $0080
022C 85 90        STA   $0090
022E 85 AD        STA   $00AD
0230 85 A5        STA   $00A5
0232 85 A7        STA   $00A7
0234 85 C4        STA   $00C4
0236 A9 00        LDA   #$00
0238 8D 00 00     STA   $0000
023B 4C 00 00     JMP   $0000
```

arises to re-initialize the BASIC start pointers. The BASIC program in listing 1 contains the parameters needed to store the machine code in user memory and provides for user input in changing the BASIC pointers.

At line 10 through 30, the machine code program is stored in user memory beginning at hex 0222 or decimal 546. The machine code is stored in the BASIC program in DATA statements at lines 100 through 130. These data are READ and POKEd into memory with the FOR... NEXT loop at lines 10 through 30. The remainder of the BASIC program simply obtains the operator's input for a new BASIC start address. This start address is obtained at line 50 and stored in the "A" variable. At line 60 and 70, this new address data is stored or POKEd into the machine code areas at 0223 or 547 decimal and 023A or 570 decimal. The USR vector is set at line 80 to point to the machine code routine beginning at 0222 hex or 546 decimal. Line 90 is a statement using the USR function of BASIC. This statement causes a jump through the USR vector to 0222 hex and executes the machine code routine.

When the program is run, the pointers will be changed to reflect the new start address. When the machine code program has reset the pointers, it jumps to BASIC warm start at hex 0000 or decimal 0. The C1P responds with OK. To set up the new BASIC work space, simply type NEW and a carriage return.

Once the BASIC program in listing 1 has been keyed into the C1P or other Ohio Scientific computer, you should SAVE the program on cassette tape for later use. This cassette program can be loaded into any relocated BASIC program space, as can any SAVED BASIC program. The Ohio Scientific SAVE and LOAD cassette commands can be used regardless of where you have relocated your BASIC program workspace.

In conclusion, I believe this information will help owners and users of Ohio Scientific computers with BASIC in ROM to better understand how the Ohio Scientific Microsoft BASIC and the OSI monitor are used. Good luck.

MICRO

Cassette I/O for SYM BASIC

Expand the capabilities of SYM BASIC with this cassette I/O handler. Access your cassette as a data file—not just as a place to save programs.

Nicholas J. Vrtis
5863 Pinetree S.E.
Kentwood, MI 49508

CASSETTE I/O FOR SYM BASIC

One of the features I missed when I started working with BASIC on the SYM was the lack of any program input or output capability to the cassette. It seemed a shame to use up a lot of memory with DATA statements which were only used once to initialize an array. They use up a lot of memory fast, since they don't get compressed much. For any type of inquiry program, even a lot of memory isn't enough. Ten minutes of cassette will hold around 90,000 characters, more than the total 6502 address space.

One Method

I considered two methods of getting the data to and from the cassette. The first was to take the data directly from BASIC's variable area. It wasn't hard to snoop around BASIC enough to find out how and where it kept variables. The usage in BASIC would have been via two USR functions; one to read, and one to write. The only argument would have been a string containing the variable name. The assembler routine to write to tape would have been almost trivial after the variable data area was located. The SYM monitor routines only need a starting and ending address. This method also had the advantage of being able to save a large array in one write to the cassette, with no data conversion. It would, however, have been a very inefficient way to handle a non-dimensioned variable (at the rate of four bytes at a time), since you do need some SYNC bytes and leader for motor startup each time you write to the tape. There are many problems that come up, moreover, when you want to read the stuff back in. The SYM routines need to know both the starting and ending address, if you want to read data back into a different area from whence it came. Due to the way BASIC dynamically allocates variables, it wouldn't be a safe bet to count on everything being in the same place as it was when written. Finding the beginning address to go back to wouldn't be too hard, since the input to the read routine would be the variable to read into, and we can find that. The ending address could again be calculated from the BASIC data, but it has to be the same length as before. For numeric data, that is easy to control with a DIM statement. Strings complicate the process even further, since their length is determined by use, not a DIM statement. Of course, one could always rewrite the SYM routines to accept starting address and length as a parameter list. I decided that there had to be an easier way.

Another Method

The second method is a little harder to use in BASIC, but turns out to have a couple of extra uses I hadn't planned on. The idea was to make the cassette respond similarly to the paper tape reader/punch on an ASR Teletype. For those unfamiliar with the beast, there are four control characters used to turn the paper tape on and off (usually called DC1 through DC4) from a remote terminal or computer. The plan was to intercept all the terminal I/O via the INVEC and OUTVEC system RAM vectors, and look for these characters to control data to or from a cassette buffer. This would mean that INPUT and PRINT statements could be used to get at the cassette. After experimenting with this for a while, I decided to make a separate routine which took care of turning cassette input and output mode on and off. This made the routines transparent to any character. (But beware, INCHR does some things after a character is obtained.) It also simplified making the output from a BASIC program suitable as input to BASIC.

The Routines Described

Since the routines intercept data via INVEC and OUTVEC, they are usable with any software written for the standard SYM, such as TINY BASIC or assembler programs. Except for the first and last buffers, filling and emptying is taken care of automatically, so you don't have to worry about how much data will fit in one buffer. In fact, data can extend from one buffer to the next and your program would never know the difference. It even turns out that the routines can be used for a "free" merge for BASIC programs. Simply LIST the programs to the cassette, and then read them back in. BASIC will think you are entering the lines from the terminal and sort the line numbers, etc.

There are four separate "entry points" in these routines. INITIT takes care of initialization and setup; CTURN handles turning cassette mode on and off, and forcing first and last buffer handling; CHRIN replaces INTCHR as the vector address for

SYM input; and finally, CHROUT handles character output in place of TOUT. INITIT and CTURN are the only two called directly by the user. The other two are used indirectly by normal input and output.

INITIT Starts the Process

INITIT really has two purposes in life. The first and foremost is to put the address of CHRIN and CHROUT into INVEC and OUTVEC in system RAM. This could have been done with a couple of SD monitor commands, but these are messy to do when you are in BASIC. The second purpose is to initialize the current FILEID to zero. Additionally, the routine also saves the current OUTVEC address in three spare system RAM locations. This is because I have a separate routine to drive my Quick Printer II, and I want to be able to use both routines. Normally, INITIT would only be called once, but it still checks to make sure it doesn't have its own address already in OUTVEC. If it saved this as the terminal output address, things could get circular.

Do's and Don'ts

For INITIT, don't have trace on while it is executing. Trace will try to use OUTVEC after it gets half changed. The program assumes that CHRIN and CHROUT start on the same page ($OF currently). If you end up relocating it to someplace where this isn't true, you will have to add some code to put in the correct starting page for one of the routines. INVEC is not saved anywhere. If you have a custom input routine, you will have to add code to save it, and change the JSR INTCHR at $F2B to use it. If you decide not to save OUTVEC, you will have to change the JSR NUVEC at $F41 and $F87 to a JSR TOUT (@ $8AA0). INITIT doesn't require any parameters, and it doesn't attempt to save or restore any registers. BASIC takes care of that for itself. You will have to watch it if you are using the routines from assembler. Obviously, it must be called before any data will go to the buffer. It may be called more than once, but any previous data in the buffer may be lost (see the discussion of input parameter value 3 for CTURN).

CTURN Controls I/O Routine

The routine CTURN is used as a common entry point for controlling the status of the I/O routines. I chose to use a single entry point for two reasons.

Listing 1

```
                    CASSETTE I/O ROUTINES FOR SYM-1 BASIC

                    SYM-1 MONITOR CALLS

1000                INTCHR  *     $8A58   TERMINAL INPUT ROUTINE
1000                LODCAS  *     $85F3   LOAD FROM CASSETTE
1000                SAVCAS  *     $87EA   WRITE TO CASSETTE
1000                GETKEY  *     $88AF   GET KEY FROM HEX PAD
1000                ERMSG   *     $8171   OUTPUT "ER XX" MESSAGE
1000                ACCESS  *     $8B86   UNPROTECT SYSTEM RAM

                    SYM-1 MONITOR VARIABLES

1000                INVEC   *     $A661   RAM INPUT VECTOR
1000                OUTVEC  *     $A664   RAM OUTPUT VECTOR
1000                NUVEC   *     $A646   UNUSED BYTES IN SYS RAM
1000                TECHO   *     $A653   TERMINAL ECHO CONTROL
1000                TAPDEL  *     $A630   TAPE LEADER TIMER
1000                P3L     *     $A64A   TAPE PARAMETER 3
1000                P3H     *     $A64B
1000                P2L     *     $A64C   PARAMETER 2
1000                P2H     *     $A64D
1000                P1L     *     $A64E   PARAMETER 1
1000                P1H     *     $A64F

0E00                        ORG   $0E00

0E00                PZBIDX  *     $00FA   BUFFER INDEX POINTER
0E00                PZCURR  *     $00FB   CURRENT BUFFER ID #
0E00                BUFLEN  *     $00FA   BUFFER LENGTH CONSTANT

                    BUFFER AREA - FIRST BYTE IS MAX INDEX VALUE

0E00 00             BUFFER  =     $00

0EFA                        ORG   $0EFA   BUFFER IS 250 BYTES

                    TURN ON/TURN OFF ROUTINES

0EFA 20 86 8B CTURN   JSR   ACCESS  UNPROTECT SYS RAM
0EFD 98               TYA           MOVE OVER & CHECK SIGN
0EFE F0 18            BEQ   NEWBF   TO FORCE NEW BUFFER
0F00 30 0D            BMI   CTOFF   TO TURN TAPE OFF
0F02 0D 53 A6         ORA   TECHO   OTHERWISE TURN IT ON
0F05 C0 03            CPY   #$03    CHECK FOR BUFFER RESET
0F07 D0 0B            BNE   TURNF   NO - STORE NEW VALUE
0F09 20 C2 0F         JSR   NWRITE  YES - SET UP POINTERS
0F0C C6 FA    ADJUST  DEC   PZBIDX  ADJUST POINTER TO START
0F0E 60               RTS

0F0F 88       CTOFF   DEY           CHANGE -1 TO FE, -2 TO FD
0F10 98               TYA           PUT IN A REGISTER

0F11 2D 53 A6         AND   TECHO   TURN OFF ECHO BIT
0F14 8D 53 A6 TURNF   STA   TECHO   STORE NEW VALUE
0F17 60               RTS           DONE

0F18 AD 53 A6 NEWBF   LDA   TECHO   READ OR WRITE?
0F1B 29 01            AND   #$01    TRY FOR READ
0F1D F0 6C            BEQ   BUFULL  NO - MUST BE A WRITE
0F1F 20 45 0F         JSR   DOREAD  YES - READ BUFFER
0F22 D0 E8            BNE   ADJUST  FIRST NOT USED YET
```

```
                    INPUT ROUTINE - INPUT RETURNED IN A REGISTER

0F24 AD 53 A6  CHRIN   LDA   TECHO    READ FROM CASSETTE?
0F27 29 01             AND   #$01
0F29 D0 04             BNE   FRMBUF   INPUT FROM BUFFER
               ** CHANGE THE FOLLOWING IF OLD INVEC WAS SAVED
0F2B 20 58 8A          JSR   INTCHR   TERMINAL I/O
0F2E 60                RTS            WILL RETURN TO $8A21

0F2F 20 CB 0F  FRMBUF  JSR   NXTBUF   GET NEXT BUFFER ADDR.
0F32 D0 05             BNE   GETIT    OK IF NOT AT END
0F34 20 45 0F          JSR   DOREAD   ELSE READ IN FIRST
0F37 B0 EB             BCS   CHRIN    READ ABORTED
0F39 BD 00 0E  GETIT   LDA   BUFFER,X GET BYTE FROM BUFFER
0F3C 2C 53 A6          BIT   TECHO    CHECK IF ECHO WANTED
0F3F 10 03             BPL   GETITR
               ** CHANGE THE FOLLOWING IF OLD OUTVEC NOT SAVED
0F41 20 46 A6          JSR   NUVEC    ECHO TO TERMINAL
0F44 60        GETITR  RTS            WILL RETURN TO $8A21

               READ NEXT BUFFER FROM CASSETTE

0F45 E6 FB     DOREAD  INC   PZCURR   BUMP TO NEXT RECORD ID
0F47 A5 FB     REREAD  LDA   PZCURR   GET NEXT ID VALUE
0F49 C9 FF             CMP   #$FF     CHECK FOR INVALID ID
0F4B D0 04             BNE   DOREAX   OK
0F4D A9 01             LDA   #$01     ELSE RESET TO 01
0F4F 85 FB             STA   PZCURR
0F51 8D 4A A6  DOREAX  STA   P3L      PUT IN PARAMETER 3
0F54 20 F3 85          JSR   LODCAS   LOAD FROM CASSETTE
0F57 90 6E             BCC   ZERIDX   CONTINUE IF LOAD OK
0F59 20 71 81          JSR   ERMSG    ELSE PRINT ERROR CODE
0F5C C9 8C             CMP   #$8C     MANUAL ABORT? (V1.1 ONLY)
0F5E F0 07             BEQ   SETOFF   YES - TURN OFF CASSETTE
0F60 20 AF 88          JSR   GETKEY   WAS FOR KEY FROM HEX PAD
0F63 C9 0D             CMP   #$0D     WAS IT A RETURN?
0F65 D0 E0             BNE   REREAD   READ AGAIN IF NOT
0F67 38        SETOFF  SEC            SET THE ERROR FLAG
0F68 A0 FF             LDY   #$FF     Y=-1 TO TURN OFF TAPE
0F6A D0 8E             BNE   CTURN    GO TURN IT OFF

               OUTPUT ROUTINE - OUTPUT CHARACTER IN A REGISTER

0F6C 48        CHROUT  PHA            SAVE CHARACTER
0F6D AD 53 A6          LDA   TECHO    CASSETTE OUTPUT?
0F70 29 02             AND   #$02
0F72 F0 12             BEQ   DOECHO   BRANCH IF NOT CASSETTE
0F74 20 CB 0F          JSR   NXTBUF   GET CASSETTE BUFFER INDEX
0F77 D0 03             BNE   STOCHR   SKIP IF BUFFER NOT FULL
0F79 20 8B 0F          JSR   BUFULL   EMPTY BUFFER FIRST
0F7C 68        STOCHR  PLA            RESTORE CHARACTER
0F7D 9D 00 0E          STA   BUFFER,X PUT INTO BUFFER
0F80 2C 53 A6          BIT   TECHO    ECHO?
0F83 10 05             BPL   NOECHO   NO
0F85 48                PHA            PUT BACK ON STACK
0F86 68        DOECHO  PLA            RESTORE CHARACTER
               ** CHANGE THE FOLLOWING IF OLD OUTVEC NOT SAVED
0F87 20 46 A6          JSR   NUVEC    OUTPUT VIA OLD OUTVEC
0F8A 60        NOECHO  RTS            WILL RETURN TO $8A52

               WRITE OUTPUT BUFFER TO CASSETTE

0F8B A9 00     BUFULL  LDA   #$00
0F8D 8D 4C A6          STA   P2L      START OF BUFFER IN P2
0F90 A2 0E             LDX   #$0E
0F92 8E 4D A6          STX   P2H
0F95 8E 4B A6          STX   P3H      AS ENDING PAGE ALSO
0F98 A2 F9             LDX   #BUFLEN-1 END ADDRESS OF BUFFER
```

First, it is easier to remember a single address instead of half a dozen, and secondly, BASIC requires at least one parameter so it can distinguish a function, so I figured I might as well use it to pass on useful information. Note that the low order of the BASIC calling parameter is passed on the Y register by BASIC. There are six input parameter values to CTURN; 0, 1, -1, 2, -2, and 3.

Four of the input values are rather trivial, and merely involve turning a bit on or off in TECHO. These bits are used to indicate that cassette input or output is active. Bit zero is used to control input mode. If it is on, input characters will be obtained from the cassette (via the buffer). Off means that input is from the terminal as normal. Bit one of TECHO performs the same function for cassette output mode. Having both bits on at the same time could cause some strange results, since there is only one buffer, and one routine would be putting characters in while the other was taking them out. The input values 1 and 2 correspond to the bit values needed to turn the appropriate mode. I decided for user simplicity to use the negative of the turn-on value to indicate a turn-off request. Note that -1 is $FF in hex, and by subtracting one we get $FE, which coincidentally has a zero in just the bit we want to turn off. (Amazing—these computers, aren't they?) The same thing happens to -2 when we subtract one from it.

The other two parameter values of zero and three require some detailed explanation, since they have different meanings depending upon whether you are in the read or the write mode. Normally, the buffer gets filled (or emptied) automatically as it gets used up. PZBIDX is an incrementing pointer that indexes to the last character used in the buffer. The first byte of the buffer area is used to hold the number of bytes in the buffer, including itself. The buffer is considered "used up" when PZBIDX equals this value. Normally, this value would be the number of bytes in the buffer. The problem occurs because a buffer is not necessarily a "logical record."

A buffer is written only when all the characters are used up normally. If my program has done all the writing I want it to do, but hasn't used up an even number of buffers, what do I do with the odd piece of data left in the buffer? Calling CTURN in output mode with a zero value will force a write of this short buffer. When we get around to reading this short buffer back in, there

has to be some way to keep track of how many bytes were used in each buffer. The short buffer may not be the last record on the tape if you decide to add data in a subsequent run of the program. This is why BUFULL transfers the current value of PZBIDX to the start of the buffer before it gets written out. That way, when it gets read back in, the maximum length is set automatically.

In the read mode, we don't have to worry about the last record being short. The write routines took care of that already. Thre is no end-of-file record or indicator maintained. It is up to the calling program to do that. The problem in the read mode occurs for the read of the first buffer. There are no logical grounds for counting on PZBIDX to point to the end of a buffer the first time I need to get a character. As a matter of fact, it stands a better chance of being zero, since INITIT leaves it that way. Since read is the opposite of write, it makes as little sense to have zero for the read mode indicate the *first* buffer instead of the *last* as it does for the write mode. Calling CTURN with an input value of zero in read mode will force a read of the first buffer from the cassette. From then on, read buffers will take care of themselves.

The final input parameter value for CTURN is three. The actual code sets PZBIDX to zero, and sets the maximum buffer value to the length of a buffer. The "logical" meaning of this depends on what you are going to do next. It was originally designed to be used before the first write to the buffer, since the above setting indicates to the write operation that the buffer has nothing in it. In the read mode, the same setting would mean that the buffer was just read in and characters could be removed.

At first, I couldn't think of why anybody would want to set the buffer full for a read without actually doing one, but then... If you write a small amount of data to the buffer (less than one full buffer), you could LOAD another program in and read the data back in by setting PZBIDX back to zero. This would be a way of getting around the fact that a LOAD clears all variables and would allow "passing" data between programs.

CHRIN Handles Character Input

All character input is handled by CHRIN. It never gets called directly by the calling program. The user still calls INCHR whenever he wants an input character. INCHR saves the caller's registers and gets to CHRIN via INVEC. Since INCHR has already saved the registers, CHRIN doesn't bother doing it. The first thing that it does do is check the low bit in TECHO to see if cassette input mode is on. If we aren't in cassette mode, the process is to get a character from the teminal by using INTCHR, and use an RTS to get back to the last part of INCHR, and from there back to the calling program.

When cassette input mode is on, a character has to be obtained from the buffer. PZBIDX is incremented and compared to the end of buffer value by a call to NXTBUF. If there is a character available, it is obtained from the buffer.

```
0F9A 8E 4A A6           STX  P3L
0F9D A5 FA              LDA  PZBIDX  GET LAST CHARACTER USED
0F9F F0 21              BEQ  NWRITE  NO WRITE IF NONE USED
0FA1 8D 00 0E           STA  BUFFER  PUT AS MAX INDEX
0FA4 A6 FB              LDX  PZCURR  GET LAST RECORD ID
0FA6 E8                 INX          INCREMENT FOR THIS RECORD
0FA7 E0 FF              CPX  #$FF    BE SURE IT IS VALID
0FA9 D0 02              BNE  BUFULX
0FAB A9 01              LDA  #$01    ELSE RESET
0FAD 86 FB     BUFULX   STX  PZCURR
0FAF 8D 4E A6           STA  P1L     USE FOR AN ID ALSO
0FB2 AD 30 A6           LDA  TAPDEL  SHORTEN TAPE LEADER
0FB5 48                 PHA          BUT SAVE OLD VALUE
0FB6 A9 02              LDA  #$02    NEED SOME TIME TO SPEED UP
0FB8 8D 30 A6           STA  TAPDEL
0FBB 20 EA 87           JSR  SAVCAS  MONITOR WRITES CASSETTE
0FBE 68                 PLA          RESTORE LEADER TIME
0FBF 8D 30 A6           STA  TAPDEL
0FC2 A9 FA     NWRITE   LDA  #BUFLEN
0FC4 8D 00 0E           STA  BUFFER
0FC7 A2 00     ZERIDX   LDX  #$00    RESET TO BUFFER START
0FC9 86 FA              STX  PZBIDX  FALL THROUGH TO NXTBUF

               GET CURRENT BUFFER INDEX AND BUMP FOR NEXT TIME

0FCB E6 FA     NXTBUF   INC  PZBIDX  INCREMENT ID
0FCD A6 FA              LDX  PZBIDX  USE AS INDEX
0FCF EC 00 0E           CPX  BUFFER  CHECK IF AT END
0FD2 60                 RTS

               ROUTINE INITIALIZATION - SET UP ADDRESSES

0FD3 20 86 8B  INITIT   JSR  ACCESS  UNPROTECT SYSTEM RAM
0FD6 A2 02              LDX  #$02    SAVE CURRENT OUTVEC
0FD8 BD 63 A6  ITRANS   LDA  OUTVEC-1,X
0FDB C9 0F              CMP  #$0F    PAGE ADDRESS OF ROUTINES
0FDD F0 06              BEQ  NOSAV   ALREADY SAVED, SO SKIP
0FDF 9D 46 A6           STA  NUVEC,X
0FE2 CA                 DEX
0FE3 10 F3              BPL  ITRANS
0FE5 A9 00     NOSAV    LDA  #$00    RESET CURRENT ID TO ZERO
0FE7 85 FB              STA  PZCURR
0FE9 A9 0F              LDA  #$0F    PAGE ADDRESS OF ROUTINES
0FEB 8D 62 A6           STA  INVEC+01
0FEE 8D 65 A6           STA  OUTVEC+01
0FF1 A9 24              LDA  #$24    ADDRESS OF CHRIN
0FF3 8D 61 A6           STA  INVEC
0FF6 A9 6C              LDA  #$6C    ADDRESS OF CHROUT
0FF8 8D 64 A6           STA  OUTVEC
0FFB D0 C5              BNE  NWRITE  SET UP INDECES

0FFD FF                 =    $FF     PATCH AREA
0FFE FF                 =    $FF
0FFF FF        ZZZEND   =    $FF     LAST BYTE OF PROGRAM
```

In order to provide the capability to echo input as INTCHR would, the ECHO bit (bit 7 of TECHO) is checked. If it is on, the input is echoed to the terminal by way of the old OUTVEC that was saved by INITIT. Either way, return is to INCHR by an RTS, the same as above.

At some point, the buffer has to run out of characters. At this point, we have physically to read another buffer from the cassette. It is really a simple matter of passing the desired FILEID to the SYM monitor and letting it do the work. The FILEID is kept in PZCURR. This value is set to zero by INITIT, and is incremented just prior to each physical read and *write* . A check is made to make sure it doesn't reach $FF, since the monitor treats that as a special ID. Zero is also avoided for the same reason. The FILEID is needed in case there is a problem reading the tape and we want to backup and retry the read. After the call to the SYM monitor, the carry is checked to see if the read was successful. If the carry is clear, the read worked, PZBIDX is set to one, a character is obtained, and return is to INCHR, the same as before.

When there is an error (the carry is set), ERMSG is called to let the monitor display the standard ERxx message. Now the problem is to determine what to do about the error. If the error code was $8C, the load was aborted by the CR key during sync search. This obviously means that the user has given up, so we might as well too. This is done by branching to CTURN with $FF in the Y register to turn off cassette mode. From then on, characters are obtained from the terminal. For other load errors, the program waits for any character to be entered on the keypad (via a call to GETKEY). At this point, the cassette remote control still is off, but you get a chance to stop it manually before trying again. If you hit any key other than CR, the program loops back and tries to read the same FILEID again. Don't forget to rewind the tape a little before putting it back in play mode. The program will continue to try reading a tape until it gets it right, or you give up.

Caution: aborting the cassette input mode does *not* stop your BASIC program. In order to stop your BASIC program after aborting the cassette input mode, enter an @ to delete the characters already received from the buffer, and then enter a carriage return. Otherwise, your program will take what it has and probably loop back and turn the cassette input mode back on. Keep cassette input mode on as little as possible in your program, and double check for syntax errors. Otherwise you may end up feeding data from the buffer to BASIC instead of to your program.

Also note that since these routines are normally called via INCHR, you won't be able to read anything you couldn't enter from the terminal. These routines will handle any bit combination, but INCHR strips parity, and upper cases everything. If you want to be able to write object code or upper and lower case and don't want to reasemble these routines, I would suggest patching out the input character echo (starting at $F3C), and replacing it with:

```
TAX          Save the input
             character for now
PLA          Discard normal
             return from stack
PLA
TXA          Return input to A
             as expected
JMP $8A3C    Go compare to CR
             & return
```

This will bypass the editing normally done by INCHR.

CHROUT Handles Character Output

The basic flow for CHROUT is the same as for CHRIN. Bit two in TECHO is checked instead of bit one, and characters are put into the buffer instead of removed, but the process is mostly the same. BUFULL is called when there is no room to put the current character into the buffer. It can also be called from CTURN to write the last buffer, so it checks to make sure there are actually some used characters in the buffer, since there is no need to write a buffer with nothing in it. A full buffer is always written, but the current value in PZBIDX is moved to the first byte of BUFFER so CHRIN will use it as the maximum buffer length when it gets read in later. The current value of PZCURR is incremented, and that is used as the FILEID for the call to the SYM monitor. Here again the values of $FF and $00 are avoided because they would pose problems reading them back in.

Note that the character is output to the terminal via the old OUTVEC value which was saved by INITIT whenever cassette output is not in effect. The amount of tape leader time is changed from the current value to two before the write, and restored after. This is an attempt to save some sync leader time. Depending on how fast your recorder starts and stops, you may want to change this. Zero will never work, since sync search on a read requires at least ten SYNC's before admitting that things are in sync. I found that a value of one did not allow enough time for my recorder to get up to speed. The value of two is marginally enough, but I am tempted to trade time for safety, and change it to three.

The maximum buffer size is 255 bytes (not 256), of which 254 are for data, and one is reserved for maximum valid buffer length. I chose 250 bytes because it made the end of the program come out about right. The page zero addresses used for PZBIDX and PZCURR are those used by the monitor as RAM input pointers for the EXECUTE command, so don't expect to use it and pick up where you left off. Change the addresses if it bothers you to use monitor locations. I chose to start the buffer at $E00 for two reasons. First, I don't use the TRIG routines much, especially in the inquiry type programs I planned for these routines. Second, the TRIG routines are self-relocating, so they will fit in front of these easily. Relocate these routines with a little bit of caution. There are a couple of places where parts of addresses are used in load immediate instructions. These have been indicated in the listing. There is no requirement that the programs in buffer occupy adjacent areas. They could be widely separated if that turns out to be convenient.

Reading and Writing Tricks

There are a couple of tricks to writing things out and reading them back into a BASIC program. The main thing is to make your output look as it would if you keyed it in. Don't forget to put commas between items, and remember that strings need quotes around them if they have commas. Probably the hardest part is remembering that BASIC only has a 72-character input buffer. If you create an output line bigger than that and then try to read it back in, you will get beeped and "EXTRA IGNORED". Setting the output line length to 72 or less doesn't help, since all that does is put a carriage return where you don't want it. In case you are interested, the line feeds which BASIC puts out after the carriage return are conveniently ignored by BASIC upon input.

Finally, the 16-bit checksum from $EFA to $FFF is $7C66. This will give you something to check against when you enter the program.

MICRO

EXCERT, INCORPORATED
*** AIM 65 ***

P/N		QTY 1-9
A65-1	AIM-65 w/1K RAM	$375
A65-4	AIM-65 w/4K RAM	$420
A65-A	Assembler ROM	$ 85
A65-B	BASIC ROM	$100
A65PL	PL/65 ROMS	$125

ACCESSORIES

P/N		QTY 1-9

Power Supplies (Fully AIM-65 Compatible, Industrial Quality Open Frame)

2PRS3	+5V at 3A, +24V at 1A w/mtg hardware, cord, etc.	$ 65
PRS4	+5V at 2A, +24V at .5A w/mtg hardwere, cord, etc.	$ 50
PRS5	+5V at 2A, +24V ±15% at .5A ±12V to ±15V at 4A.	$ 75

From The Enclosure Group

ENC1	AIM-65 case	$ 45
ENC1A	AIM-65 case w/space for one expansion board	$ 49

Cases with Power Supplies

ENC3	ENC1 w/PRS3 mounted inside	$115
ENC3A	ENC1A w/PRS3 mounted inside	$119
ENC4	ENC1 w/PRS4 mounted inside	$100
ENC4A	ENC1A w/PRS4 mounted inside	$104
ENC5	ENC1 w/PRS5 mounted inside	$125
ENC5A	ENC1A w/PRS5 mounted inside	$129

From The Computerist, Inc.

MCP1	Mother Plus ™ Dual 44 pin mother card & card cage takes MEB4 VIB1, PTC1, fully buffered, 5 expansion slots underneath the AIM	$100
MEB4	DRAM Plus ™ 16K RAM, 16K Prom sockets, 2-6522 I/O chip and programmer for 5V EPROMS	16K RAM $295 32K RAM $395
PTC1	Proto Plus ™ Prototype card same size es KIM-1, MEB4, VIB1	$45
VIB1	Video Plus ™ board with 128 char, 128 user char, up to 4K display RAM, light pen and ASCII keyboerd interface	$245
CABLE	For MEB4, VIB1, PTC1	$ 15

From Optimal Technology

ADC1	A/D eight channels, D/A 2 channels. Requires ±12V to ±15V at 100 MA & 2-I/O Ports from AIM-6522	$115
	Cable	$ 25

SPECIALS

P/N		QTY 1-9
A65-4AB	AIM-65 w/4K RAM Assembler and BASIC	$595
A65-4B	AIM-65 w/4K RAM, BASIC	$510
S/A65-1	AIM-65 w/1K RAM, w/o PRINTER, Display Keybd, User 6522	$199

P/N		QTY 1-9

From Seawall Marketing, Inc.

MCP2	Little Buffered Mother ™ Single 44 pin (KIM-4 style) mother card tekes MEB2, PGR2, PTC2 and P102. Has on board 5V regulator for AIM-65, 4 expansion slots. Routes A&E signals to duplicates on sides with 4K RAM	$199
MEB2	SEA 16™ 16K stetic RAM board takes 2114L with regulators and address switches	$280
PGR2	Programmer for 5V EPROMS with ROM firmware, regulators, low force sockets, up to 8 EPROMs simulteneously, can execute efter programming	$299
P102	Parallel I/O board with 4-6522's	$260
PTC2	Proto/Blank™ Prototype cerd that fits MCP2	$ 49
PTC2A	Proto/Pop™ with regulator, decoders, switches	$ 99

From MTU

DAC3	8 bit DAC Board	$49
FDC3	Floppy disk controller bd. & DOS, up to four 5¼" or 8" drives, double sided, double density	$595
MCP3	Card file w/4 slot expansion mother bd. w/keybd. brackets	$ 85
MEB3A	16K DRAM Board, low power	$298
PI03	24K PROM, 4-8 bit I/O ports w/RS-232 port to 4800 bps. PROM Programmer	$295
VIB3	8K DRAM Bd. low power w/composite video out in 200 lines 320 dot/line format	$240

All MTU Software Available for these Products.

Miscellaneous

TPT2	Approved Thermal Paper Tape 5/165' rolls	$ 10
MEM6	6/2114 RAM Chips	$ 45
CAS1	Audio Cassette Recorder	$ 40
2716	16K5V EPROM	$ 20
A65-P	Printer	$ 75
A65-DM	Display Module	$ 30

All AIM-65 Spare Parts Are Avelleble.

ASSEMBLED & TESTED SYSTEMS

We specialize in assembled and tested systems made from the above items. Normally, the price will be the total of the items, plus $5 for handling, shipping is extra on all C.O.D.'s or invoiced orders. Please call or write for exact prices or if questions arise. Six month warranty on all systems.

Higher quantities quoted upon request. COD's accepted, shipping will be added. Add $5 for shipping, insurance and handling on prepaid orders. Minnesota residents add 4% sales tax. Prices subject to change without notice

Mail Check or Money Order To:
EXCERT, INC.
Educational Computer Division
P.O. BOX 8600
WHITE BEAR LAKE, MN. 55110
612-426-4114

Multiplying on the 6502

The 6502 processor may not provide for fast multiplication—but here are five routines to speed up multiplication on any 6502 system.

Brooke W. Boering
Vagabond Enterprises
1300 E. Algonquin 3G
Schaumburg, IL 60195

The search for the ultimate multiply routine seems never-ending. For those APPLE owners who have been using the monitor 'MUL' routine in order to get assembly language efficiency, and for others just looking for some fast 6502 multiply code, the following should be of interest.

Fast multiplication has always been very desirable. Over the years, processors have often provided this as a hardware command. Not so, however, on our favorite 6502. The new 16-bitters and some hybred 8-bitters now provide it, and execution speed is on the order of 10 to 30 microseconds.

Taking the APPLE 'MUL' routine as a starting point (fig. 1), we can calculate execution speeds based on the number of bits 'on' in the multiplier as follows:

Multi. bits 'on'	Speed
16 (maxi.)	1511 μs.
8 (aver.)	1223 μs.
0 (min.)	935 μs.

In addition to execution time, we must add the overhead of pre-loading or zeroing working bytes and/or registers, and the entry instruction (JSR). This is significant only when such overhead is materially different when comparing alternate techniques. In the case of the

Machine Language Subroutines

Figure 1

```
MULTIPLYING ON THE 6502
BY BROOKE W. BOERING

               ORG    $1000

WORKING BYTES FOR ALL ROUTINES

ACL      *     $0050
ACH      *     ACL+01
XTNDL    *     $0052
XTNDH    *     XTNDL+01
AUXL     *     $0054
AUXH     *     AUXL+01
AUX2L    *     AUXL+02
AUX2H    *     AUXH+02

APPLE "MUL" ROUTINE
   16X16 MULTIPLY
ON ENTRY: MULTIPLICAND IN AUXL, AUXH
          MULTIPLIER IN ACL, ACH
          XTNDL, XTNDH MUST BE ZERO

A, X, Y NOT SAVED

ON EXIT: 32-BIT RESULT IN XTNDH, XTNDL,
         ACH, ACL

1000 A0 10    MUL    LDY   #$10     INDEX FOR 16 BITS
1002 A5 50    MUL2   LDA   ACL      ACL * AUX + XTND
1004 4A              LSR   A          TO AC, XTND
1005 90 0C           BCC   MUL4     IF NO CARRY,
1007 18              CLC             NO PARTIAL PRODUCT
1008 A2 FE           LDX   #$FE
100A B5 54    MUL3   LDA   AUXL,X   ADD MULTIPLICAND
100C 75 56           ADC   AUX2L,X    TO PARTIAL PRODUCT
100E 95 54           STA   AUXL,X
1010 E8              INX
1011 D0 F7           BNE   MUL3
1013 A2 03    MUL4   LDX   #$03
1015 76 50    MUL5   ROR   ACL,X
1017 CA              DEX
1018 10 FB           BPL   MUL5
101A 88              DEY
101B D0 E5           BNE   MUL2
101D 60              RTS
```

(continued)

'MUL' routine, this amounts to an additional 39 microseconds due primarily to the requirements of working byte pre-loading.

An examination of the actual code used in 'MUL' reveals that a very bad tradeoff was taken, which increased execution time by a whopping 70-75%! It did save 2 bytes of code, however. By replacing both internal loops with 'in line' code, this flaw is remedied (fig. 2).

The revised routine cannot be 'stuffed into' the ROM monitor and, therefore, must be executed somewhere in our own code. Once this negative factor is accepted, it is feasible to examine other possible improvements in both speed and convenience.

Fig. 3 shows a repackaging of the revised 'MUL' routine. It has allowed us to incorporate two improvements. First is the removal of some pre-entry overhead by having the caller simply load 2 registers, and completing the storing internally in this new 'MUL16X' routine. Secondly, we have 'frontended' MUL16X to test for a multiplier of 8 bits or less.

The technique of loading registers with arguments rather than requiring the caller to perform the STORE code does not improve overall execution time. Its primary virtues are reduced code in the caller's pre-entry sequences and improved flexibility within the service routine.

The idea of testing for an 8-bit (or less) multiplier is to be able to effect some improvement in execution speed whenever that condition is true. Such a test performed at the start of MUL16X allows for dynamic variation in multiplier length. There are other times, however, when the programmer knows for certain that the multiplier is limited to 8 bits or less. To cover both cases and continue to provide exit conditions common to 'MUL', a new routine, MUL816 is shown in fig. 4.

(continued)

Figure 2

```
                REVISED "MUL" ROUTINE
                  16X16 MULTIPLY

                NOTE: ENTRY & EXIT CONDITIONS SAME AS "MUL"

101E A0 10      RMUL    LDY   #$10     16-BIT MULTIPLIER
1020 A5 50      RMUL2   LDA   ACL      AC * AUX + XTND
1022 4A                 LSR   A         TO AC, XTND
1023 90 0D              BCC   RMUL4    IF NO CARRY
1025 A5 52              LDA   XTNDL    ADD MULTIPLICAND
1027 18                 CLC            TO PARTIAL PRODUCT
1028 65 54              ADC   AUXL
102A 85 52              STA   XTNDL
102C A5 53              LDA   XTNDH
102E 65 55              ADC   AUXH
1030 85 53              STA   XTNDH
1032 66 53      RMUL4   ROR   XTNDH    SHIFT INTERIM RESULT
1034 66 52              ROR   XTNDL
1036 66 50              ROR   ACL
1038 88                 DEY            DECREMENT LOOP COUNTER
1039 D0 E5              BNE   RMUL2
103B 60                 RTS            EXIT
```

Figure 3

```
                M U L  6 X
                16X16 MULTIPLY

                ON ENTRY: MULTIPLIER IN ACL, ACH
                          MULTIPLICAND LOW IN Y
                          MUTIPLICAND HIGH IN X REG.

                ON EXIT: 32-BIT RESULT IN XTNDH, XTNDL,
                         ACH, ACL

103C A5 51      MUL16X  LDA   ACH      8-BIT MULTIPLIER?
103E F0 2F              BEQ   MUL5X    YES, GOTO COMMON
1040 A9 10              LDA   #$10     SET 16-BIT MULTIPLY
                (FALL THROUGH TO COMMON CODE)
1042 84 54      MULTI   STY   AUXL     SAVE MULTIPLICAND LO
1044 86 55              STX   AUXH     SAVE MULTIPLICAND HI
1046 A8                 TAY            MULTIPLIER COUNT TO Y
1047 A9 00              LDA   #$00
1049 85 52              STA   XTNDL    ZERO PARTIALS
104B 85 53              STA   XTNDH
104D A5 50      MUL3X   LDA   ACL      ACL * AUX + XTND
104F 4A                 LSR   A         TO AC, XTND
1050 A5 52              LDA   XTNDL    ADD MULTIPLICAND
1052 18                 CLC            TO PARTIAL PRODUCT
1053 65 54              ADC   AUXL
1055 85 52              STA   XTNDL
1057 A5 53              LDA   XTNDH
1059 65 55              ADC   AUXH
105B 85 53              STA   XTNDH
105D 66 53      MUL4X   ROR   XTNDH    SHIFT INTERIM RESULT
105F 66 52              ROR   XTNDL
1061 66 51              ROR   ACH
1063 66 50              ROR   ACL
1065 88                 DEY            DECREMENT COUNT
1066 D0 E5              BNE   MUL3X    LOOP TIL DONE
1068 60                 RTS
```

Figure 4

```
                    M U L 8 1 6
                    8X16 MULTIPLY

                    ON ENTRY: MULTIPLICAND LOW IN Y,
                              MULTIPLICAND HIGH IN X,
                              MULTIPLIER (8 BITS) IN A REG.

                    ON EXIT: 24-BIT RESULT IN Y, A, X, AND
                             XTNDL, ACH, ACL (XTNDH+0)

1069 85 50          MUL816  STA   ACL      SAVE MULTIPLIER LO
106B A9 00                  LDA   #$00
106D 85 51                  STA   ACH      ZERO MULTIPLIER HI
106F A9 08          MUL5X   LDA   #$08     SET 8-BIT MULTIPLY
1071 20 42 10               JSR   MULTI    DO COMMON CODE

                    TRANSFER ANSWER FROM HIGH THREE BYTES TO
                     LOW THREE BYTES AND LOAD EXIT REGISTERS
1074 A4 53                  LDY   XTNDH
1076 A9 00                  LDA   #$00
1078 85 53                  STA   XTNDH
107A A5 52                  LDA   XTNDL
107C 84 52                  STY   XTNDL
107E A6 51                  LDX   ACH
1080 85 51                  STA   ACH
1082 86 50                  STX   ACL
1084 60                     RTS

                    M U L 8 H I
                    SPECIAL 8X16 MULTIPLY
                    8-BIT MULTIPLIER IS HIGH BYTE, NOT LOW

                    ON ENTRY: MULTIPLICAND LOW IN Y,
                              MULTIPLICAND HIGH IN X,
                              8-BIT MULTIPLIER-HIGH IN ACC.

                    ON EXIT: 24-BIT RESULT IN XTNDH, XTNDL,
                             ACH, (ACL=0)

1085 85 50          MUL8HI  STA   ACL      ZERO MULTIPLIER
1087 A9 00                  LDA   #$00
1089 85 51                  STA   ACH      ZERO MULTIPLIER HIGH
108B A9 08                  LDA   #$08     SET 8-BIT MULTIPLIER
108D 4C 42 10               JMP   MULTI    DO COMMON CODE
                    (RTS)

                    M U L 8 X 8
                    8X8 MULTIPLY

                    ON ENTRY: MULTIPLICAND IN Y,
                              MULTIPLIER IN A REG.

                    ON EXIT: RESULT HIGH IN Y,
                             RESULT LOW IN A REG.

                    TIMING: 212 MICROSECONDS MAXIMUM
                            180 MICROSECONDS MINIMUM
                            192 MICROSECONDS AVERAGE
```

MUL816 is entered when the multiplier is known to be 8 bits or less. It provides an execution time improvement on the order of about 37% over the use of the revised 'MUL' routine.

Shown along with MUL816 is a special 'MUL8HI' routine, which is actually an alternate entrance to MUL816 whenever the 8-bit multiplier is known to be the HI order byte of a normally 16-bit multiplication. Its primary virtue lies in combining the efficiency of an 8X16 operation with the same exit protocols of MUL16X, since both must supply 32-bit answers.

By this time it should be obvious that we can improve things even more if we address ourselves to the 8-bit by 8-bit multiply as a separate matter. Note that we are building a 'family' of routines which we can count on to execute at top speed with ease of use.

Fig. 5 shows this last member of our family, MUL8X8. Its features are several. There is no pre-storing to working bytes. Both the preload by caller and the result are register-oriented, since the multiplier and multiplicand are both 8-bit, while the answer is limited to 16 bits. It requires only 25 bytes of code.

The use of MUL8X8, whenever both the values are 8-bit limited, results in a further improvement in execution time of about 50%, when compared to using MUL816. Compared to 'RMUL', the speed increases by some 69% while a whopping 82% improvement is seen over APPLE's 'MUL' routine in its ROM version!

For those interested, the execution times quoted here are based on average 'bits on in the multiplier' of 8 for 16-bit executions and 3 for 8-bit guys. Both maximum and minimum calculations were also performed and changed the percentages only 2% or 3% for comparable multipliers.

Here is a rough breakdown of average execution times:

Routine	16-Bit	8-Bit
'MUL'	1262 μs	1082 μs
RMUL	726 μs	631 μs
MUL16X	725 μs	391 μs
MUL816		393 μs
MUL8X8		192 μs

(continued)

(continued)

Figure 5

```
1090               ANSLO    *     $0050
1090               PLIER    *     $0051
1090               CAND     *     $0052

1090 85 51         MUL8X8   STA   PLIER     SAVE (MULTI)PLIER
1092 84 52                  STY   CAND      SAVE (MULTIPLI)CAND
1094 A9 00                  LDA   #$00      RA=RESULT HIGH
1096 A0 08                  LDY   #$08      SET 8-BIT COUNTER
1098 46 51         MUL1Y    LSR   PLIER     TEST NEXT BIT
109A 90 03                  BCC   MUL2Y     IF OFF, GO ROUND
109C 18                     CLC
109D 65 52                  ADC   CAND      IF ON, ADD
109F 6A            MUL2Y    ROR   A         SHIFT ANSWER 1 BIT
10A0 66 50                  ROR   ANSLO     *
10A2 88                     DEY             DECREMENT POS. COUNTER
10A3 D0 F3                  BNE   MUL1Y     LOOP UNTIL DONE 8 BITS
10A5 A8                     TAY             Y=RESULT HIGH BYTE
10A6 A5 50                  LDA   ANSLO     A=RESULT LOW BYTE
10A8 60                     RTS
```

Summary

Note that although these routines are presented as improvements on the APPLE ROM routine, they are usable on all 6502 systems as they are free-standing.

While 192 microseconds for an 8X8 multiply may not be spectacular compared to 15 or 20 for a hardware command, it should be close to the fastest available on our 1-MHz 6502. For those who don't see any need for this speed, there is at least one chip maker that thinks super fast multiplication will be in high demand. Advanced Micro Devices has a new chip, the AM25S558 [the 'RABBIT'] which attacks an 8X8 multiply in, get this, 45 nsec! Furthermore, the latched version supports cascading 4 of these chips to do 16X16 operations in 100 nsec.

MICRO

MICRO Club Circuit

Mike Rowe
Club Circuit
P.O. Box 6502
Chelmsford, MA 01824

The following club announcements are presented in zip code order.

PET User Group
New England Computer Society
Meets every third Wednesday at the Mitre Corp. Cafeteria [Rte. 62 at Rte 3, Burlington, MA.]. William C. Ekle is President for approximately 30 members. Address any correspondence to:

19 Winchester Drive
Merrimack, NH 03054

Purpose: *"To further the knowledge of the PET and to share information."*

New York City Area User's Group
Meets the first Thursday of each month at 7:00 p.m. David Gillette is the president, and the club now has 40 members. For more information, please contact:

Mike Bassman
39-65 52nd St.
Woodside, NY 11377

To enlighten OSI users as to what can be done on an OSI.

APPLE Power Users Group
Meets the second or third Wednesday of every month (7:00 p.m.) at: Syosset High School, Syosset, Long Island, New York. To contact the club concerning membership, library program exchanges, newsletter exchanges, etc., please write to:

Apple Power, c/o M . Lack
8 Division Street
Holtsville, Long Island,
New York 11742

Jim Lyons is the president of our club. Membership is now at 110 and is rapidly expanding. We have a bi-monthly newsletter, "The Pits," which has been a big success. Yearly dues are only $20 and include a free subscription to our newsletter, computer hardware and software discounts, feature demonstrations and presentations at all meetings and an extensive program library. We now offer 4 different library paks. We welcome new members and encourage information, program and newsletter exchanges with all other Apple users groups.

UAUG
Upstate Apple User Group
Meets on the third Thursday of the month at 7:30 p.m. at the Upstate Computer Shop (629 French Road, Campus Plaza, New Hartford, N.Y.). Bill Etter is President for this group of 20. Contact Tony Violente, Public Relations at:

629 French Road
Campus Plaza
New Hartford, N.Y. 13413

"Aim to offer support to new Apple owners and support local school projects. Aid in software and hardware problem solution."

Delmarva Computer Club
Meets for a business meeting on the 1st Wednesday of each month and an informal meeting on the 3rd Wednesday of each month, both at 7:30 p.m. Address correspondence to Jean Trafford, Secretary, at:

P.O.Box 36
Wallops Island, VA 23337

"Primary objectives: Aiding the handicapped, bringing computer awareness to the community, providing the opportunity to the community to use and program computers. Non-profit organization — marketing a manual alphabet tutorial program for the PET computer, with all proceeds going to fund club projects."

OSI — MUG
Ohio Scientific Michigan User's Group
This is a recently formed group with a membership of approximately 130 people. They would like to have their presence known to other user groups and to people interested in becoming members. Contact:

Ralph V. Johnson, Sec.
OSI — MUG
3247 Lakewood Avenue
Ann Arbor, MI 48105

SLACC
St. Louis Area Computer Club
Meets on the first Thursday of every month at 7:00 p.m. Membership numbers around 120. Dennis W. Jolly is spokesperson and President. Meetings are held at the Thornhill Branch of the St. Louis County Public Library at Willowyck and Fee Fee Road. Contact:

SLACC
P.O.Box 28924
St. Louis, MO 63132

"Promotes the understanding and growth of microcomputing through group activities. We are a processor non-specific club. We are a non-profit registered organization with a monthly newsletter. Meetings are open to the public and membership is not required in order to attend."

OKC Apple User Group
Meets on the first and the third Tuesday of each month at 7:00 p.m. at various computer stores in Oklahoma City area. Andy Gin is the President. Contact:

OKC Apple User Group
Secretary
Greenbriar Digital Resources
P.O.Box 1857
Edmond, OK 73034

"For the benefit of Apple users, owners, and anyone having an interest in personal computing. Member of IAC. Exchange information and ideas, publish 'OKC Apple Times' newsletter, club library, etc."

The APPLE CORPS of Dallas, Texas
The Apple Business and Personal Applications group of the APPLE CORPS meets on the second Saturday of each month at 10:00 a.m. For information on the meeting and place contact:

Bob Matzinger
P.O. Box 13446
Arlington, TX 76013

To develop, review and discuss applications of, and software for the Apple Computer related to business and personal data processing. We are NOT a software exchange group!

APPLE B.U.G. (Bakersfield Users Group)
Meets the first Friday of each month, at 7:30 p.m. For location of meetings, please check with the Computer Warehouse. President is Gary F. Atchison, and membership stands at 25. For more information please contact:

Bob Geisler
20333 Old Town Road
Star Rt. #2
Tehachapi, CA 93561

To enjoy our Apple Computers.

ATARI Computer Enthusiasts (ACE)
Meets every second Wednesday of each month at different members' homes. Stacy Goff is the president and membership totals 20. Address correspondence to:

M.R. Dunn
c/o ACE
3662 Vine Maple
Eugene, OR 97405

To publish newsletter, teach about the ATARI, develop and exchange programs, help in fund-raising for non-profit organizations, aid handicapped members.

MICRO offers a free one year subscription to all clubs registered with us. For registration form write to:

MICRO Club Circuit
Box 6502
Chelmsford, MA 01824

MICRO

Microbes and Updates

Mike Rowe
Microbes & Updates
P.O. Box 6502
Chelmsford, MA 01824

Edward F. Kurtz, Jr. from Northampton, Massachusetts tells us: I have been using the information in the article "Applesoft Floating Point Routines" by R.M. Mottola, MICRO August 1980 (27:53). In the process I discovered that the subroutine FPDIV2 at $EA60 does not seem to pay any attention to the sign of the numerator or denominator! I am therefore using only the subroutine FPDIV at $EA66 for all division operations, and it seems to work properly.

Earl Morris of Midland, Michigan sent this update: I have converted Tiny Pilot (MICRO 16:41) to run on my OSI system. The additions to Tiny Pilot by Bob Applegate (MICRO 27:21) make this an interesting interpreter. However, Applegate's random number function will occasionally return invalid numbers. This is caused by illegal BCD numbers being stored in the seed locations. The 6502 CPU when in the decimal mode cannot handle all combinations of illegal BCD numbers. Consider the decimal addition problem OF + 05. The correct answer (decimal) is 20. However the 6502 cannot carry a "2" into the tens column. The best answer it can give is 1A. This is a correct answer, but not correct BCD format. This answer returned to PILOT will give an invalid number.

Below is the correction to the random function. The "1" is added last instead of first.

Old Code	New Code
SED	SED
SEC	CLC
LDA $D5	LDA $D5
ADC $D8	ADC $D8
ADC $D9	ADC $D9
STA $D4	ADC #$01
	STA $D4

Adding the "1" last gives the CPU opportunity to convert the result into proper BCD format.

C.R. MacCluer of East Lansing, Michigan has some updates to his article, "Satellite Tracking with the AIM 65" which appeared in the August 1980 MICRO (27:13).

The text in the third column of page 13 should read:

```
100 A=7281:  E=0:  P=103:
    K=99: W=0
300 FOR T=0 TO P
```

And in the listing:

Line No.	Was	Should Be
110	E=67627	E=.67627
340	–M1)/:01–	–M1)/(01–
350	102–8	10 –8
460	THEN 470 NEXT T	THEN 480
470		NEXT T
4020	108–8	10 –8

From Shankill, County Dublin, Ireland, Charles H. Putney writes: I read with interest the article in August 1980 MICRO (27:17) by Frank Chipchase on RENUMBER. I prefer a cruder approach of making RENUMBER easier to use. The program listed below creates a text file which when EXEC'ed saves the current program, runs RENUMBER, and restores the program being worked on. Be sure to put RENO on the same diskette as RENUMBER. During a session when RENUMBER is needed just EXEC RENO.

```
PROGRAM: CREATE RENO
 10 D$=" ":REM CNTL D
 20 PRINT D$; "OPEN RENO"
 30 PRINT D$;"WRITE RENO"
 40 PRINT "SAVE RENOFILE"
 50 PRINT "NEW"
 60 PRINT "RUN RENUMBER"
 70 PRINT "N" :REM ANSWER
    TO INSTRUCTIONS
    NEEDED?
 80 PRINT "LOAD RENOFILE"
 90 PRINT "DELETE
    RENOFILE"
100 PRINT "PRINT"
    ;CHR$(34);"RENUMBER
    LOADED";CHR$(34)
110 PRINT D$;"CLOSE"
120 END
```

Gary M. Ganger of New Carlisle, Ohio caught this error: I would like to point out an error on page 5 of #27. In the description of the front cover the Space War Game was first on the DEC PDP-15, not on a PDP-1. (There never was a PDP-1.)

Pete Cook of Mesa, Arizona writes: A few Microbes have become apparent in the article "Creating Shape Tables, Improved!" (28:7).

1. In figure 3, the zero should have one more dot placed in its center. The dots are so close together that you can't tell the difference on the screen, but the missing dot becomes very noticeable if you plot the character display on a printer. Three bytes in the shape table must be changed. Run the program, end it, then type the following:

```
POKE 20044, 44
POKE 20045, 12
POKE 20046, 4
BSAVE SHAPEFILE NUMERALS,
   A20000, L188
```

2. Some applications require the use of a blank shape, such as using the space bar within a different alphabet set. To allow this, change line 3100 to 3102 and add new line 3100:

```
3100 IF PEEK (ADDR – 1) = 0
     THEN POKE ADDR, 0:
     ADDR = ADDR + 1:
     GOTO 3170
```

To make a blank shape, simply type "Q". The program will place a zero in the shape table, followed by a zero end-of-record mark.

3. Line 2460 should read "GOTO 2520" rather than "GOTO 2570". This caused the cursor from the last shape to appear in the next shape, and caused some dots to appear in the upper right corner of the screen.

4. Line 4510 should read "XDRAW" instead of "DRAW". This one requires some explanation. The article states it is possible to erase a plotted point by plotting another point over the top of it. This only works if you use "XDRAW" to draw the shape. As the shape is drawn, the point will be plotted in white the first time and in black the second time, thus erasing it.

Don't forget to save the program again after all the changes have been entered.

MICRO

MICRO

Software Catalog: XXVII

Name: **Rental Manager**
System: APPLE II or APPLE II + or Language System
Memory: 48 K
Language: Applesoft
Hardware: APPLE II, Disk II w/2 drives, Printer
Description: A total system for rental property management. It handles accounts receiveable, accounts payable and the general ledger. Maintains the chart of accounts and buildings, listings for present and future tenants, and will print reports and notices for "problem tenants." Automatically posts entries between systems and creates back-up copies of information entered. Includes a separate instructional version and full user documentation.
Price: $695.00
Author: **Bill Tesnow** (a property manager)
Available: **Blue Lakes Computing**
438 N. Frances
Madison, WI 53703

Name: **R & G DATABASE**
System: APPLE II and APPLE II Plus
Language: Applesoft
Hardware: APPLE II with Disk, Optional Line Printer
Description: A keyed random access filing system, providing rapid access to items on file under full user control. The suite of more than five programs is run by explicit questions and simple keyboard responses. This allows even the non-programmer to set up a professional filing system. The report generator works with screen or line printer. Reports are selective and result fields can be generated, as can new headers. Print and screen format are under full control of the user. It prompts the user when he is likely to make a mistake. An unusual feature is the easy to use comprehensive data validation specified by the user and then used by the system for all further file inputs. State Configuration. (No. of Drives etc.)
Price: $85.00 plus shipping
Includes: Manual and Disc
Author: **John Robinson**
Available: **R & G MICRO's,**
550 Midgeland Rd,
Blackpool, Lancashire,
England

Name: **Restaurant Evaluation**
System: Apple II, Apple Plus
Memory: 16K
Language: Applesoft II
Hardware: Optional: Disk II, printer
Description: Evaluates potential restaurant/nite club sites and thereby reduces the margin of risk involved in purchasing a new or existing business. All the necessary percentages and formulas are programmed to evaluate whether a potential site will be profitable or not. The program is also structured for use by present restaurateurs to evaluate whether or not their present business is operating at cost and profit efficiency. Calculates monthly gross, computes monthly loan notes (or mortgages), and reports weekly, monthly and annual net profit/loss in dollar amounts and percentages. All rights reserved.
Price: $19.95 + $2.00 (P&H) First Class Mail, Check or Money Order.
Available: **Mind Machine, Inc.**
31 Woodhollow Lane
Huntington, N.Y. 11743

Name: **Wind Energy Calculations**
System: Apple II or Apple II Plus
Memory: 16K
Language: Applesoft II
Hardware: Printer optional
Description: This program allows the prospective wind-charger buyer, builder, or experimenter to familiarize himself with the theory behind obtaining power from the wind. Simply input values for various parameters and choose how they are to be incremented for the calculations. Learn what propeller parameters are compatible with any size generator, what effect an increase or decrease in height has on power output, and what to expect from any combination of system size, efficiency, and wind velocity. Parameters that can be calculated include size, RPM, and tip-speed-ratio of propellers, useful watts available, kilowatt-hours per day output, and more.
Price: $9.50 postpaid on cassette
Available: **Charles O'Neill**
3-C Liberty Lane
Elk City, Okla. 73644

Name: **Hi-Res Shape Encode**
System: Apple II
Memory: 16K
Language: Integer BASIC
Description: This is a 16K Integer BASIC program that uses standard 40X40 graphics to encode several shapes into a Shape Table for display in graphics. The program displays both plot and non-plot moves. After encoding, non-plot moves may be erased. The bytes of each shape are stored in a Shape Table for writing on cassette tape. Shapes are changed by tracing forward or backward along a shape's moves. Text and graphic displays allow moves to be erased, changed, or inserted. The program has options to delete, copy, move, replace, or swap entire shapes in a Shape Table. Documentation includes features, cassette operation, new shapes, reading and writing Shape Tables, modifying shapes and tables, examples, and program structure.
Price: $7.50, check or money order
Includes: Program and shape table on cassette, documentation
Author: **Harry L. Pruetz**
Available: **MICROSPAN Software**
709 Caldwell St.
Yoakum, TX 77995

Name: **The Aliveness Life Dynamic**
System: Apple II
Memory: 48K
Language: Applesoft, Machine
Hardware: Apple II, Disk II
Description: Adds up to a long and intense look at, and "workshop" on, all the major barriers to *aliveness*. Deals with life-awareness, and increasing the understanding of the interrelationships between feelings, aliveness, rational vs. irrational, neurosis and awareness. Most of the disk centers on three unique games (in Hi-Res with shape tables): *The Primal Oil Fields, The Keys to Awareness,* and *Rationality!* You'll love them as games, learn from them as enlightening experiences.
Price: $15.95
Includes: Disk, game card
Available: **Avant-Garde Creations**
P.O. Box 30161 MCC
Eugene, OR 97403

Name: **Empire of the Stars**
System: OSI C1P
Memory: 8K
Language: 8K Basic-in-ROM
Hardware: C1P or Superboard
Description: Grand scale strategic simulation of the rise to power of 1 to 4 interstellar empires. Players wield fleets of up to thousands of ships in an attempt to rule the galaxy. Captured and colonized worlds produce new ships.
Price: $9.95
Author: **Gorple the Rigellian**
Available: **Horizon Computing**
P.O. Box 479
Mendham, NJ 07945

Name: **Diskolog**
System APPLE II
Memory: 48K
Language: Applesoft
Hardware: APPLE II, Disk II
Description: A utility program that will alphabetically catalog a large number of programs. Features include: FIND A DISK —locates disk name by inputting program name or characters in name; CATALOG A DISK —lists all programs on a disk; LIST ALL PROGRAMS —lists all programs on file and indicates disk name; ADD A DISK —adds a new disk to file (using screen read feature); DELETE A DISK —deletes a disk from file; LIST BY TYPE —lists all programs by type (A, B, I, T); RENAME A DISK —changes name of disk in file; and LIST ALL DISKS —lists disks in file and indicates number of sectors remaining.
Price: $14.95
Includes: Disk, Instructions
Available: **Computek**
28278 Enderly Street
Canyon Country, CA 91351

Name: **DATA HANDLER**
System: APPLE II, APPLE II Plus, PET
Memory: 7K
Language: Applesoft, PET BASIC
Hardware: Floppy disk drive
Description: A disk oriented data manager software package. Easy data file creation and powerful record handling. User can sort, merge, add, delete, update, view, print, write data files and more. Special features include mass updating and sorting by fields. Code is easily modified. Excellent for office use!
Price: $25.00 postpaid
Includes: Software on floppy disk, documentation, and example applications.
Author: **Rick Keck**
Available: **Business Computer Services Co.**
9020 Eby
Overland Park, KS 66212

Name: **OPTIMIZED EP—2A SOFTWARE**
System: Any 6502
Memory: 1.25K
Language: Assembly
Hardware: Standard EP-2A
Description: Turns the EP—2A into a professional quality EPROM programmer. Five commands are available: ERASE verify, PROGRAM, PROGRAM verify, COPY PROM to RAM, EXIT. Full address prompting is performed; appropriate messages are printed (e.g., PROGRAMMING...). Extensive error checking is performed. Cassettes available for AIM, KIM, and SYM systems; others may load object (or source) from listing provided. Loads at $0200. Specify system!!!
Price: $19.95 Listing and instructions
$2.00 Cassette (ASK FAMILY ONLY),$ 2.00 Custom Assembly
Author: **Jeff Holtzman**
Available: **Jeff Holtzman**
6820 Delmar—203
St. Louis, MO 63130

Name: **C RAE**
System: Apple II or Apple Plus
Memory: 48K
Language: Applesoft on ROM
Hardware: 3.2 DOS & Disk
Description: Co-resident Applesoft Editor for Applesoft programs. Perform global change/finds to most anything in your Applesoft program, quote a range of lines, a stop-list that produces a fault optimized listing, dump, a very fast renumber, APPEND, AUTOLINE Numbering. All commands invoked with one key command and needs to be loaded only once while you are developing and running your program.
Price: $14.95 on disk
Available: **Highlands Computer Services**
14422 SE 132nd
Renton, WA 98055

These listings are free. In order to obtain a free listing, however, software producers or distributors must follow the format used here. The description part of the listing may NOT exceed 12 typeset lines.

Listings are published in the order in which they are received. However, only one entry per company is accepted for publication in any single month. When more than one entry is submitted, the company submitting the entry must establish the relative priorities. If the software product or its price change after an entry is submitted, it is the responsibility of the company to notify MICRO prior to publication.

Name: **BrownPak 1 Diskette**
System: APPLE II
Memory: 16K-48K
Language: Applesoft in ROM
Hardware: Disk II is preferable
Description: A diskette of utility routines. Machine language routines include print using, sort, packing and unpacking, data, and a special input command. Applesoft BASIC Routines include auto diskette menu, disk-free utility, Hi-Res shape utilities and a general input routine.
Price: $39.95
Author: **Donald Brown**
Available: **The Computer Emporium**
3711 Douglas
Des Moines, IA 50310

Name: **APARTMENT MANAGER**
System: APPLE II or APPLE II Plus
Memory 48K (Firmware Card if APPLE II)
Language: Applesoft II and Assembly
Hardware: 2 Disk Drives, 132 column printer
Description: Maintains financial and managerial data for up to 6 separate complexes; each complex can contain a maximum of 120 units (user determined). Maintains MTS, YTD rental income for all tenants on file. Calculates security deposit interest. Generates operating statements, and rental totals as well as much more.
Price: $325 includes manual
Author: **Gary E. Haffer**
Available: **Software Technology for Computers**
P.O. Box 428
Belmont, MA 02178

Name: **Video Games 1**
System: OSI, C2, C4, C8 BASIC-in-ROM
Memory: 8K
Language: BASIC
Hardware: None special
Description: Video Games 1 consists of three games; Head-On, Tank Battle, Trap! Head-On is an arcade-style game for one. You must try to avoid a head-on crash by changing lanes. Tank Battle is a tank game for two to four players. Trap! is a blockade-style game, with enhancements for one or two. Color and sound for machines so equipped.
Copies: Just released
Price: $15 on cassette tape, ppd.
Author: **Mike Bassman**
Available: **Orion Software Associates**
147 Main Street
Ossining, N.Y. 10562

****

OHIO
SCIENTIFIC

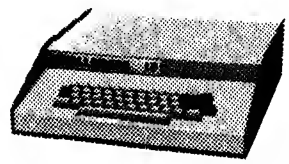
SUPERBOARD II $ 279.00
610 BOARD 8K $ 279.00
620 BOARD $ 99.00
(OSI BUS FOR 610)
630 BOARD $ 225.00
(ADDS COLOR ETC.)
C4P series 2 $ 859.00
C8P-DF-32K $2939.00
AC20 COLOR MONITOR
10 in.-NOT A TV-$399.00

VISA, MASTERCHARGE-O.K.-
****
SHIPPING AND INS. CHARGE
ADDED TO CREDIT ORDERS.
CHECKS MUST CLEAR BANK
BEFORE SHIPMENT:NO COD.
****
E&I TECHNICAL SERVICE
5300 PARIS GRAVEL RD
HANNIBAL, MO.63401
314-248-0084
****

OHIO SCIENTIFIC

FREE ATV MICROVERTER with

Challenger 1P



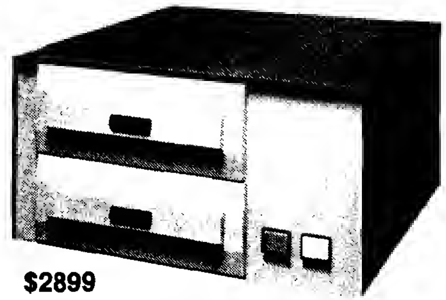
COMPUTER SHOP

Boston
590 Comm. Ave.
(across from B.U.)
247-0700

Union N.H.
Rte 16B
603-473-2323

Cambridge
288 Norfolk St.
(near M.I.T.)
661-2670

C2OEM-
The Best Buy
In 8" Disk Systems



$2899

The C2-OEM cabinet can be table top, rack mounted or incorporated in a matching desk which will accommodate a CRT terminal and printer. Industry standard parts provide very reliable operation and easy service. And best of all, the C2-OEM can make use of most of the business application software and accessories for the popular, premium performance Ohio Scientific Challenger III.

****SPECIAL INTRODUCTORY OFFER****

Programmable Character Generator Board **$89.95**

You can use OSI's characters or you can make your own. Imagine you can now do true high resolution graphics 512 x 256 dots in the 64 x 32 screen format. And all under your control!

Other mods available — send for catalog.

SOFTWARE (with Documentation)

PC Chess V1.9 $14.95
Play Chess against your computer!

Helicopter Pilot: (64 CHR Video Only) $ 8.95
An Excellent Graphics Program!

Golf Challenger $14.95
From 1 to 4 players. Play a round of golf on your 18 hole golf course. One of the best programs I have ever seen! You can even design your own course. Comes with full documentation (14 pages).

Two Very Intricate Simulations!

Wild Weasel II: You operate a Sam Missile base during a Nuclear War. Not as easy as you think! You must operate in a three dimensional environment.

Failsafe II: The shoe is on the other foot! Here you are in the attacking bomber and you must penetrate deep into enemy territory. Can you survive? An extremely complex electronic warfare simulation! SPECIAL: both for 19.95

Hardware: C1P Video Mod: Makes your 600 Video every bit as good as the 4P and 8P. Gives 32/64 CHR/Line with guardbands 1 and 2 Mhz. CPU clock with 300, 600 and 1200 baud for Serial Port. Complete Plans $19.95

KIT(Hardware and Software) $39.95
Installed: 32CHR — $79.95, 64CHR-$89.95
Extra K of Video RAM for 64CHR not included!

C1P Sound Effects Board: Completely programmable! For the discriminating hobbiest, the best board on the market for creating sound and music. Can be interrupt driven so that you can use it for gaming purposes. Has on board audio amp, 16 bit interval timer, 128 Bytes of RAM and two 8 bit parallel I/O Ports.

Assembled and tested $89.95 Bare Board $39.95
Both Include Prog. Manual and Sample Software.

C1P Hi Speed Cassette Kit: Gives a reliable 300, 600, and 1200 Baud. No symmetry adjustments — the ideal fix for OSI's cassette interface. Easily implemented in 30 minutes. Will save you time and money even the first night you use it! $12.95

Many, many more. Send for Catalog with free program (Hard Copy) and BASIC Memory Map. $1.00. Two locations to serve you:

Progressive Computing
3336 Avondale Court, Windsor, Ontario
Canada, N9E 1X6
(519) 969-2500
or
3281 Countryside Circle, Pontiac TWP, MI 48057
(313) 373-0468

VISA **MASTER CHARGE**

In the December issue of the Ohio Scientific Small Systems Journal we are presenting a brief description of the new Vocalizer software and three, user contributed, game programs.

If you are interested in contributing software or other articles to the Small Systems Journal contact:

Small Systems Journal
c/o Ohio Scientific, Inc.
1333 S. Chillicothe Rd.
Aurora, Ohio 44202

Ohio Scientific manufactures two products which support speech synthesis, the CA-14A and CA-15V. The CA-15V is the Universal Telephone Interface which was described in the Small Systems Journal of the June 1980 MICRO. The CA-14A contains virtually the same speech output circuitry but has a non-populated voice input area designed for experimental use.

Both the CA-14A and the CA-15V generate artificial speech with a VOTRAX® speech synthesis module. The VOTRAX speaks words on a phoneme-by-phoneme basis so it is possible to reproduce nearly every word of the English language.

Phonemes can be considered as the basic building blocks of the spoken word. A few of the phonetic codes used by the VOTRAX are:

Phoneme	Typical Useage
TH	three
UH	but
EH	ten
ER	her
U	two
O	note

GREAT PYRAMID

```
10 PRINT:PRINT:PRINT
15 GOTO3010
20 PRINTTAB(16)"How many levels in the pyramid";
25 INPUTA:PRINT:PRINT
30 IFA<13THEN50
40 PRINT"The number of levels specified exceeds the display"
41 PRINT"capabilities of the video monitor, sorry. ":PRINT
45 GOTO20
50 FORX=1TO32:PRINT:NEXT
55 POKE9770,0
60 DIMA(A,3)
70 FORX=1TOA:A(X,2)=X:NEXT
75 FORX=1TOA:W=W+X:NEXT:GOTO2000
80 REM
90 PRINT"-----Which site do you wish to move a level from ?"
100 POKEB,128:C=PEEK(B):IFC<16THEN100
105 IFC=16THEN90
110 IFC=128THENF=1
120 IFC=64THENF=2
130 IFC=32THENF=3
135 FF=PEEK(F(F))
140 FORX=1TO5:POKEF(F),161:FORY=1TO20:NEXTY:POKEF(F),FF
141 FORY=1TO20:NEXTY,X
142 PRINTSPC(63):PRINT
150 PRINT"-----Which site do you wish to move a level to ? "
160 POKEB,128:C=PEEK(B):IFC<16THEN160
165 IFC=16THEN90
170 IFC=128THENT=1
180 IFC=64THENT=2
190 IFC=32THENT=3
194 FF=PEEK(F(T))
195 FORX=1TO5:POKEF(T),161:FORY=1TO20:NEXTY:POKEF(T),FF
196 FORY=1TO20:NEXTY,X
200 IFF=TTHEN1000
210 FORX=1TOA:IFA(X,F)<>0THENF1=X:GOTO230
220 NEXTX:GOTO1000
230 FORX=1TOA:IFA(X,T)<>0THENT1=X:GOTO260
240 T1=X:NEXT
245 REM
250 A(T1,T)=A(F1,F):A(F1,F)=0:GOSUB400:GOSUB300:GOTO80
260 IFA(F1,F)>A(T1,T)THEN1000
270 IFT1-1=0THEN1000
280 T1=T1-1:GOTO245
300 FORX=1TOA:A1=A1+A(X,1):A2=A2+A(X,3):NEXT
310 IFA1<>WANDA2<>WTHENA1=0:A2=0:RETURN
340 POKE9770,64:FORX=1TO16:PRINT:NEXT:PRINTTAB(27)"Game over. "
350 FORX=27TO35:PRINTTAB(X)CHR$(135);:NEXT
360 PRINT:PRINT:PRINT
370 PRINTTAB(5)"MINIMUM SCORE"TAB(28)"........"TAB(55)INT(2↑A-1)
380 PRINT
390 PRINTTAB(5)"YOUR SCORE"TAB(28)"........"TAB(55)O
391 FORX=1TO11:PRINT:NEXT
392 INPUT"DO YOU WANT TO PLAY AGAIN";A$
393 IFLEFT$(A$,1)="Y"THENRUN
394 FORX=1TO16:PRINT:NEXT
395 PRINTTAB(26)"OK  BYE NOW"
396 FORX=1TO11:PRINT:NEXT
399 RUN"BEXEC*
400 E=F(F)-128-64*A:FORX=1TOA:IFPEEK(E)<>32THEN420
410 E=E+64:NEXT
420 E=E-10:FORX=ETOE+20:POKEX,32:NEXT
430 E=F(T)-128-64*A
440 FORX=1TOA:IFPEEK(E)=164THEN460
450 E=E+64:NEXTX
460 E=E-64-A(T1,T)
470 FORX=ETOE+2*A(T1,T):POKEX,164:NEXTX
475 O=O+1
480 RETURN
1000 PRINT"------------------------- Illegal move wise guy !!!"
1010 FORX=1TO1150:NEXT:PRINTSPC(63);A$;
1020 PRINT:GOTO90
2000 REM
2010 F(1)=54286:F(2)=55007:F(3)=54320
2015 POKEF(1),49:POKEF(2),50:POKEF(3),51
2020 J=F(2)-128
2030 FORX=ATO1STEP-1
2040 LL=J-A(X,2)
2050 FORY=LLTOLL+A(X,2)*2:POKEY,164
2051 NEXTY
2060 J=J-64:NEXTX
2070 POKE2073,96
2080 B=57088:POKEB,128
```

```
3000 GOTO80
3010 PRINTTAB(23)"The Great Pyrahio"
3020 FORX=23TO39:PRINTTAB(X)CHR$(135);:NEXTX:PRINT:PRINT
3030 FORX=1TO23
3040 READA$:B=32-INT(LEN(A$)/2):PRINTTAB(B)A$
3050 NEXT:PRINT
3060 GOTO20
4000 DATAThe year is 1000 B. C.
4010 DATA"You are in charge of over 100000 hen, your job"
4020 DATA was to build The Great Pyramid of Egypt.
4030 DATAUnfortunately you built it on the wrong site and now have
4040 DATAonly one year to move it to one of two alternative
4050 DATA"sites.  Sounos simple, well it would be except for"
4060 DATAa few catches.,,
4070 DATAFirst the fyrahid has been built in a series of levels
4080 DATAeach of which hust be hoved as a single unit.,
4090 DATASecond each level is a different siie and because of
5000 DATAEgyptian law a large section of a pyrahio may not
5010 DATAbe placed on top of a shall section.,
5020 DATA"Finally, when moving the pyrahio any levels rehoveo
5030 DATAcan only be set oown on the sacred ground of the
5040 DATAthree sacred sites.  That is one of the two
5050 DATA alternative sites or the site the fyrahio is presently
5060 DATAon. Luckily you get to chose the nuhber of
5070 DATAlevels that have been cohpleted.
5080 DATA"Also, the maxihum height for a pyaahid is 12 levels
```

ROAD RACE

```
 5 FORX=1TO32:PRINT:NEXT:GOTO1000
 10 FORX=1TO32:PRINT:NEXT
 20 PRINT"CAR #1:"TAB(27)"ROAD RACE"TAB(54)"CAR #2:"
 30 PRINT"  WINS:"TAB(56)"WINS:"
 40 PRINT:PRINT:PRINT
 50 PRINT"            ...........      ......         ..........."
 60 PRINT"          .......... ......  ..........  ................."
 70 FORX=6TO57:PRINTTAB(X)". ";:NEXT:PRINT"  "
 B0 FORX=5TO58:PRINTTAB(X)". ";:NEXT:PRINT:Y=Y+1:IFY<4THEN80
 B5 A=7:B=56
 90 FORX=ATOB:PRINTTAB(X)". ";:NEXT:PRINT"  "
 100 A=A+2:B=B-2:Z=Z+1:IFZ<3THEN90
 110 FORX=1TO10:PRINT:NEXT
 111 FORX=54424TO54439:POKEX,32:NEXTX
 112 POKE54361,32:POKE54374,32:FORX=54365TO54370:POKEX,32:NEXTX
 113 POKE54303,32:POKE54304,32
 120 P=20*64:FORX=BTO18:FORY=0TO63
 130 C=53248+X*64+Y
 140 IFPEEK(C)=32THENPOKEC,161:POKEC+P,161:GOTO160
 150 POKEC,32:POKEP+C,32
 160 NEXTY:P=P-128:NEXTX
 161 FORX=54566TO54931STEP64:POKEX,1B5:NEXT
 170 A=54900:8=22:D=54772:E=254:POKEA,B:POKED,E
 180 G=57088:POKE2073,96
 190 FORX=1TO8:READH(X):NEXTX:DATA-64,-63,1,65,64,63,-1,-65
 192 FORX=1TO10
 193 C=INT(RND(WS)*1300+100)+53700:IFPEEK(C)<>32THEN193
 195 POKEC,227:POKEC+1,228
 196 NEXTX
 200 POKEG,8
 210 IFPEEK(G)=128THENB=B-1:IFB<16THENB=23
 220 IFPEEK(G)=64THENB=B+1:IFB>23THENB=16
 230 POKEA,B
 240 POKEG,4
 250 IFPEEK(G)=4THENE=E-1:IFE<248THENE=255
 260 IFPEEK(G)=2THENE=E+1:IFE>255THENE=248
 265 IFPEEK(A+H(B-15))=185ANDR=1THEN29B
 270 POKED,E
 275 R=0
 280 A=A+H(B-15):IFPEEK(A)=185THENPOKEA-H(B-15),32:GOTO400
 290 IFPEEK(A)<>32THENA=A-H(B-15):GOTO29B
 295 POKEA-H(B-15),32:POKEA,B
 298 IFPEEK(D+H(E-247))=1B5ANDS=1THEN200
 299 S=0
 300 D=D+H(E-247):IFPEEK(D)=185THENPOKED-H(E-247),32:GOTO500
 310 IFPEEK(D)<>32THEND=D-H(E-247):GOTO200
 320 POKED-H(E-247),32:POKED,E
 360 GOTO200
 400 C1=C1+1:A$=STR$(C1):A1=LEN(A$)
 405 Q=H(B-15):IFQ=1ORQ=65ORQ=-63THENC1=C1-1:R=1:A=A-Q:GOTO200
 410 FORX=1TOA1:POKE53510+X,ASC(MID$(A$,X,1)):NEXT
 415 A=A+2*(H(B-15))
 420 POKEA-Q,32:IFPEEK(A)<>32THENA=A-Q
 425 POKEA,B
```

As you would expect, building words with phonetic codes is very versatile. However, it can also be tedious and time consuming. For example, consider the word "did". Its phonetic construction with the VOTRAX is D,I,D. That was pretty simple, but look at the construction of another short word "has"—H,AE,I3,Z. This is a little more difficult to code but still sort of obvious. As the words become longer, the coding becomes much less obvious—"average" AE1,I3,V,R,I2,D,J.

Ohio Scientific has developed two new software packages to help minimize the difficulties associated with phonetic coding. These are Vocalizer I and Vocalizer II.

Both Vocalizer I and Vocalizer II operate by automatic translation of English text into phonetic code. This is accomplished by examining each individual word as it is encountered, dividing the word into utterable phonemes, and finally, speaking each phoneme through the VOTRAX.

The actual division of words is done by a set of 327 different rules. The bulk of these rules is based on the *Naval Algorithm* developed at the U.S. Naval Research Laboratory in Washington, D.C. by Honey S. Elovitz et. al. in 1975. This algorithm results in the correct pronunciation of approximately 90 percent of all words (97 percent of all phonemes) in an average sample of English text. The remaining words typically have single errors which are easily corrected by the listener.

An example of one of the translation rules is that the letter "E", when used as the final letter of a word, is not pronounced if it is preceded by zero or more consonants and one or more vowels. The words "hoe", "parade", "picture", and "bee" illustrate this rule.

Although the *Naval Algorithm* is obviously quite complex, Ohio Scientific's implementation of it is relatively compact. It requires approximately 4K bytes of memory to store and interpret the rules. The algorithm is also quite fast, translating in less than "real time". This means that a new phoneme is ready before the VOTRAX has finished pronouncing the previous one. This is true even with "older" Ohio Scientific systems operating with a one megahertz CPU clock.

The Vocalizer software can be used either as a phonetic code development tool, or as an actual output "device".

In developmental applications, words or phrases can be presented as input to the system, which responds by outputting the proper phonetic codes in a written form. These phonemes can then be modified or optimized at a later time.

The more common use of the Vocalizer is as an additional standard output for Ohio Scientific BASIC. When used in this fashion, all of BASIC's normal output is translated to phonetic code and spoken by the VOTRAX.

For example, this means that instead of having prompts printed at the terminal, they may instead be spoken. In practice, the normal BASIC line

```
100 PRINT"ENTER THE X
    COORDINATE";
```

is changed to

```
100 PRINT#6,CHR$(1),
    ENTER THE X
    COORDINATE";
```

and the VOTRAX verbally requests the entry of coordinate data.

```
430 IFC1=WSTHEN700
440 GOTO200
500 C2=C2+1:D$=STR$(C2):D1=LEN(D$)
501 Q=H(E-247)
502 IFQ=1ORQ=65ORQ=-63THENC2=C2-1:S=1:D=D-Q:GOTO200
505 FORX=1TOD1:POKE53564+X,ASC(MID$(D$,X,1)):NEXT
510 D=D+2*(H(E-247))
520 POKED-Q,32:IFPEEK(D)<>32THEND=D-Q
525 POKED,E
530 IFC2=WSTHEN710
540 GOTO200
700 W$="CAR #1 WINS":GOTO720
710 W$="CAR #2 WINS"
720 FORX=1TO11:POKE54169+X,ASC(MID$(W$,X,1)):NEXT
730 W$="Type  <1> to continue  <2> to stop"
740 FORX=1TO34:POKEX+54221,ASC(MID$(W$,X,X)):NEXT
745 POKE57088,128
750 GO=PEEK(57088):IFGO=128THEN900
760 IFGO=64THENRUN"BEXEC*
770 GOTO750
800 FORX=1TO10:PRINT:NEXT:PRINTTAB(27)"Road Race"
805 FORX=27TO35:PRINTTAB(X)CHR$(145);:NEXT:PRINT:PRINT
810 PRINT"The object of the game is to complete N number of"
815 PRINT"laps of the track before your opponent."
816 PRINT"N is specified by the user."
820 PRINT:PRINT"The player on the left controls this:";
821 POKE55105+POS(0),22:PRINT
825 PRINT"The player on the left controls this:";
826 POKE55105+POS(0),254:PRINT
830 PRINT:PRINT:PRINT
835 PRINT"To control the ";:POKE55105+POS(0),22:PRINT
840 PRINT"*****  <S> ---------- to turn counter clockwise"
845 PRINT"*****  <D> ---------- to turn clockwise"
850 PRINT:PRINT
855 PRINT"To control the ";:POKE55105+POS(0),254:PRINT
860 PRINT"*****  <M> ---------- to turn counter clockwise"
865 PRINT"*****  <,> ---------- to turn clockwise"
870 FORX=1TO5:PRINT:NEXT
875 WS=INT(WS)
880 INPUT"How many laps are required to win the race";WS
885 WS=INT(WS+.5)
890 GOTO10
900 IFC1>C2THENW1=W1+1:GOTO910
905 W2=W2+1
910 REM
915 A$=RIGHT$(STR$(W1),1):D$=RIGHT$(STR$(W2),1)
920 C1=ASC(A$):C2=ASC(D$)
925 POKE53576,C1:POKE53630,C2
930 C1=0:C2=0:RESTORE:POKEA,32:POKED,32
935 FORX=54169TO54181:POKEX,32:NEXT
936 FORX=1TO3:POKE53564+X,32:POKE53510+X,32:NEXT
940 FORX=54221TO54256:POKEX,32:NEXT:IFW1=5ORW2=5THEN950
945 GOTO170
950 FORX=1TO32:PRINT:NEXT
955 PRINT"CAR #1 SCORED "W1" WINS "
960 PRINT"CAR #2 SCORED "W2" WINS "
965 FORX=1TO12:PRINT:NEXT
970 PRINT "HIT <1> TO PLAY AGAIN OR <2> TO STOP"
980 PRINT "FOLLOW YOUR ANSWER WITH A <RETURN>":INPUTA$
990 IFA$="1"THENRUN
995 RUN"BEXEC*
1000 INPUT"Do you need the instructions (Y/N)";A$
1005 IFLEFT$(A$,1)="Y"THEN800
1010 GOTO870
```

CONCENTRATION 2

```
1 DIMPL$(2):FORX=1TO32:PRINT:NEXT
2 FORX=1TO2:PRINT"Player #"X"enter name followed by <RETURN>";
3 INPUTA$:A$=LEFT$(A$,6):PL$(X)=A$
4 NEXTX:FORX=1TO5:PRINT:NEXT
7 DIMA(7),C(8,8),F(2),L(2),O(2),P(8,8),R(2),S(7),T(4),U(9),W(2),D$(3)
10 B$=CHR$(161)+CHR$(161)+CHR$(161)+"    ":A=48:POKE2073,96
20 FORX=1TO4:C$=C$+B$:NEXTX:FORX=1TO4:FORY=1TO3:PRINT"    "+C$:NEXTY
50 FORY=1TO3:PRINT"          "+C$:NEXTY:NEXTX
60 PRINT:PRINT
65 FORT=1TO9:READU(T):NEXTT:DATA1,-63,-64,-65,-1,65,64,63,0
66 FORX=1TO7:READS(X):NEXTX:DATA128,64,32,16,8,4,2
70 FORX=53379TO53402:POKEX,187:POKEX+1600,187:NEXT
82 G=53666:FORY=1TO10:READA$
84 FORX=1TOLEN(A$):POKEG+X,ASC(MID$(A$,X,1)):NEXTX:G=G+64:NEXTY
```

```
90 DATA THE OBJECT OF GAME,IS TO GET THE MOST PAIRS
91 DATA THIS IS DONE BY,CHOOSING TWO CARDS.  IF THEY
92 DATA MATCH THEN YOU GET,1 POINT AND ANOTHER GUESS
93 DATA GUESS THE NUMBER FIRST ,THEN GUESS THE LETTER
94 DATA GAME ENDS WHEN ALL PAIRS,HAVE BEEN FOUND.
100 FORX=53378TO55006STEP64:POKEX,187:POKEX+25,187:NEXTX
130 FORX=53505TO54913STEP192:A=A+1:POKEX,A:NEXT
140 FORX=55044TO55067STEP3:POKEX,A+9:A=A+1:NEXT
160 FORX=1TO8:READQ:FORY=1TO8:P(X,Y)=Q:Q=Q+3:NEXTY:Q=0:NEXTX
180 FORX=1TO8:FORY=1TO8STEP2
190 Q=INT(RND(5)*255+1):C(X,Y)=Q:C(X,Y+1)=Q:NEXTY:NEXTX
220 FORX=1TO75:FORA=1TO4:A(A)=INT(RND(9)*8+1):NEXTA
230 A=C(A(1),A(2)):B=C(A(3),A(4)):C(A(1),A(2))=B:C(A(3),A(4))=A
241 NEXTX:G=54306
242 FORJ=GTO54331:POKEJ,145:POKEJ+128,144:POKEJ+320,144:NEXTJ
243 FORJ=GTO54626STEP64:POKEJ,147:POKEJ+8,149:POKEJ+25,146:NEXTJ
244 POKE54373,78:POKE54374,65:POKE54375,77:POKE54376,69
245 FORX=GTO54626STEP64:POKEX+17,149:NEXTX:GOTO700
260 H=57088
270 FORI=1TO2
300 FORM=1TO2:A=0:B=0
301 POKE9770,0
302 PRINT"-----"PL$(I)" CHOOSE NUMBER FOR CARD #"M
310 POKEH,128
320 FORO=1TO7:IFPEEK(H)=S(O)THENA=O:GOTO340
325 NEXTO
330 POKEH,64:IFPEEK(H)=128THENA=8:GOTO340
335 GOTO310
340 PRINT"-----"PL$(I)" CHOOSE LETTER FOR CARD #"M
341 POKEH,2:IFPEEK(H)=64THENB=1:GOTO380
345 POKEH,8:IFPEEK(H)=64THENB=4:GOTO380
350 IFPEEK(H)=32THENB=6:GOTO380
355 IFPEEK(H)=16THENB=7:GOTO380
360 IFPEEK(H)=8THENB=8:GOTO380
361 POKEH,4:IFPEEK(H)=64THENB=3:GOTO380
362 IFPEEK(H)=16THENB=2:GOTO380
365 POKEH,16:IFPEEK(H)=64THENB=5:GOTO380
370 GOTO341
380 FORX=54050TO54079:POKEX,32:NEXT
400 F(M)=PEEK(P(A,B))
410 IFPEEK(P(A,B))=187THEN750
500 FORX=1TO9:POKEP(A,B)+U(X),C(A,B):NEXTX
505 L(M)=C(A,B)
510 IFA=A1ANDB=B1THEN750
515 FORK=1TO1000:NEXTK
523 A1=A:B1=B
525 R(M)=P(A,B)
526 A=0:B=0
527 PRINTSPC(60):PRINT
528 FORX=1TO100:NEXTX
530 NEXTM
532 PRINTSPC(60):PRINT
534 A1=0:B1=0
539 IFL(1)=L(2)THEN600
565 FORX=1TO2:FORY=1TO9
570 POKER(X)+U(Y),F(X)
575 NEXTY:NEXTX
590 NEXTI
595 GOTO270
600 S(B)=S(B)+1
610 FORX=1TO2:FORY=1TO9:POKER(X)+U(Y),187:NEXTY:NEXTX
620 P=P+1
625 O(I)=O(I)+1
627 G=54572
630 FORX=1TO2:A$=STR$(O(X)):FORY=1TOLEN(A$)
635 POKEG+Y,ASC(MID$(A$,Y,1))
640 NEXTY:G=G+9:NEXTX
650 IFP<32THEN300
660 POKE9770,64
690 RUN"BEXEC*
700 G=54379:FORX=1TO2:FORY=1TOLEN(PL$(X))
701 POKEG+Y,ASC(MID$(PL$(X),Y,1)):NEXTY:G=G+9:NEXTX
705 G=54564:D$="SCORE":FORY=1TO5:POKEG+Y,ASC(MID$(D$,Y,1)):NEXTY
720 G=53666:FORY=1TO10:FORX=1TO27:POKEG+X,32:NEXTX:G=G+64:NEXTY
730 GOTO260
750 FORX=1TO32:PRINT"-";:FORY=1TO20:NEXTY,X
752 PRINT" ILLEGAL CHOICE -*- TRY AGAIN
755 FORX=1TO2000:NEXTX
756 FORX=1TO60:PRINT" ";:FORY=1TO25:NEXTY,X:PRINTZ$;
757 PRINT
760 GOTO302
1000 DATA53508,53700,53892,54084,54276,54468,54660,54852
```

Obviously, since the PRINT statement is used, information and data may be used for speaking results as well as prompting for input.

Vocalizer II software has an additional feature: it can search files for vocabulary matches, as well as generating speech via the *Naval Algorithm*. Also, the user may add special words and abbreviations to the disk-based dictionary.

An example of this is the BASIC reserved word REM. When REM is encountered by Vocalizer II, it is automatically pronounced as "remark". Clearly, this could not be done if only translation by the rules were employed.

The Vocalizer software systems are an extremely important addition to the Ohio Scientific software catalog. For computers that require speech output facilities, the Vocalizer is nearly indispensable.

Vocalizer I is available under OS-65D V3.2 and OS-65U Level I. Vocalizer II is available for both LEVEL I and LEVEL III OS-65U.

BASIC Games

BASIC game programs have always been popular in the Small Systems Journal. This month we are including three contributed games:

Great Pyramid
Road Race
Concentration 2

All of the games include their own instructions.

Two of the three games are two-player games. Great Pyramid is a single player game similar to the classic Towers of Hanoi puzzle.

All of the games operate under OS-65D V3 on 4P and 8P disk-based systems.

For use--

HI-RES GRAPHICS FOR THE APPLE II

PADDLE-GRAPHICS/TABLET GRAPHICS

The most powerful graphic development system available. Upper/lower case text may be drawn in any size, direction or color. Pictures may be sketched and filled in with any of 21 HI-RES colors (must be seen to believe!!) A shape may be constructed automatically from any object appearing on the HI-RES screen.

Paddle-graphics is for use with the standard game paddles distributed with your APPLE and TABLET-GRAPHICS is for use with APPLES' GRAPHICS TABLET.

Paddle and Tablet-Graphics are available now at your local computer store and require 48K Applesoft in rom and a disk drive. To order directly send $39.95 for Paddle-Graphics or $49.95 for Tablet-Graphics to:

On-Line Systems
36575 Mudge Ranch Road
Coarsegold, CA 93614
209-683-6858
VISA, MST CHG, COD, CHECK ACCEPTED

Look for Hi-Res Football coming soon

SOFTWARE FOR OSI

Video Games 1 .... $15.
Three Games. Head-On is like the popular arcade game. Tank Battle is a tank game for two to four. Trap! is an enhanced blockade style game.

Video Games 2 .... $15.
Three games. Gremlin Hunt is an arcade-style game for one to three. Gunfight is a duel of mobile artillery. Indy is a race game for one or two.

Adventure: Marooned in Space .... $12.
An adventure that runs in 8K! Save your ship and yourself from destruction.

Dungeon Chase .... $10.
A real-time video game where you explore a twenty level dungeon.

Board Games 1 .... $15.
Two games. Mini-gomoku is a machine language version of five stones gomoku. Cubic is a 3-D tic-tac-toe game. Both with graphics.

Disassembler .... $12.
Use this to look at the ROMs in your machine to see what makes BASIC tick. Reconstruct the assembler source code of machine language programs to understand how they work. OUr disassembler outputs unique suffixes which identify the addressing mode being used, no other program has this!

Super! Biorhythms .... $15.
A sophisticated biorhythm program with many unique features.

C1 Shorthand .... $12.
Use only two keys to enter any one of the BASIC commands or keywords. Saves much typing when entering programs. Written in machine language.

For all BASIC-in-ROM systems. Selected programs available on disk. Color and sound on video games.
Send for free catalog lising many more programs.

Orion Software Associates
147 Main Street
Ossining, NY 10562

BAP$ SOFTWARE
8K Programs for OSI C2/4

1980 INCOME TAX ESTIMATES (1040). Tax tables written into program. Just input the figures and program displays taxes due or refund. Even computes carry-forward cap. gains or losses. Excellent year-round spot-check tax program. $19.95

STOCK CHARTING. Let your computer draw your charts. Displays daily highs, lows, closes and volume. $15.95

PERSONAL FINANCE PACKAGE. 4 programs — tax info. file, budgeting, mileage and current income/payables. $17.95

UNDERSTANDING FINANCIAL STATEMENTS. Four 8-K programs. Great tutorial for intro. to financial statements. $24.95

Add $1.50 for shipping. Add $4 for disk.

BAP$ SOFTWARE

6221 Richmond Ave; Suite 220
Houston, Tx. 77057

WP-6502
a very fine word processor

for OHIO SCIENTIFIC

Tape (C1,C2,C4) $75	8" 65D & 65U $125
5" Disk (C1,C2,C4) ... $75	Descriptive Brochure ..... FREE
8" Disk for 65D $75	

Dwo Quong Fok Lok Sow
23 East 20th Street
New York City, New York 10003
(212)685-2188

MICRO

Up from the Basements

By Jeffery Beamsley

One of the unique features of Ohio Scientific 6502-based computers is the bus (or mother board) used to interface the various boards that make up the system. Many of my customers and dealers have asked me from time to time why they don't see more independent vendors providing compatible hardware for Ohio Scientific systems. The bus, its history, and its features may give us the answer to those and other interesting questions.

The 48-pin Ohio Scientific bus is really a model of efficiency. It is made up of four 12-pin Molex-type connectors. Of these 48 pins, only 42 are defined, leaving 6 available for future expansion. The defined pins on the bus include 20 address lines, 6 power lines, 8 data lines, and 8 control lines. The bus supports distributed, fully-regulated DC power. The placement of the power lines causes dead shorts on the bus for any board improperly inserted. The Ohio Scientific bus was one of the first microprocessor busses to support bi-directional data lines. It is passively terminated and probably has a bandwidth of 5 MHz. It is very inexpensive as far as bus structures are concerned and is classed by Ohio Scientific as proprietary.

The bus arrived very early in Ohio Scientific's growth, about four years ago. It was probably laid out by Mike Cheiky, vice president and motive force behind Ohio Scientific, and it is indicative of much of his thinking at that time. Molex connectors are cheap and don't use gold. They also make both the mother board and system boards somewhat easier to manufacture. There may have even been some thought about compatibility with Southwest Tech, which at that time, with its 6800-based micro, was a real contender for dominance in the microcomputer marketplace. The actual placement of signals on the bus may have been originally intended to facilitate the construction of the 420 memory board that Ohio Scientific was making at the same time. There is no other logical reason for why the address lines are so jumbled on the bus.

In a classic case of shortsightedness, the 420 memory board was designed to be as single-sided as possible and use no sockets as a cost-saving factor. This meant that the signal paths to the 2102 memories used on the board all had to occur on the foil side of the board, to permit easy desoldering of the IC's. This understandably put some rather severe restrictions on the layout of the address lines. So, if you can't design the memory board to fit the bus, why not design the bus to fit the memory board? I've always felt that foresight is a function of circumstance much more than good planning, anyway.

The circumstance Ohio Scientific found itself in, with the wane of Southwest Tech, was that it was the only "non-standard" manufacturer in an S-100 world. Rather than fold against the competition, Ohio Scientific made the sort of marketing decisions that has made it the force in the marketplace that it is today. The company decided to sell completely assembled and tested systems at a price competitive with what a hobbyist would have to pay to assemble one from various S-100 kit vendors.

As the first company to enter the microcomputer system market, Ohio Scientific turned its lack of S-100 compatibility into a marketing advantage by creating a captive market of system users completely reliant on Ohio Scientific for products. This choice, though, forced Ohio Scientific into a position where it was required to provide this market of users and dealers—in order to retain their interest—with at least the same variety of machines and accessories that were available to the S-100 user. One of the results, four years later, is a company—the only company—that covers the entire spectrum of microcomputers, from the very low-cost personal machines through the multi-user, hard disk-based computers.

Ohio Scientific is understandably protective of the captive, profitable market it has created. Because the bus is not copyrighted, second source vendors have made attempts to market Ohio Scientific compatible products. Those efforts, though, have met with considerable resistance from the "factory." It is certainly within a manufacturer's rights to be concerned about the effect of "foreign" boards in its systems. Ohio Scientific, however, seems to take its concern well beyond that of well-intentioned warnings to its users.

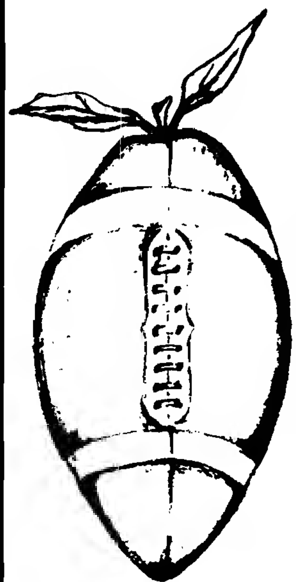
If nothing else, the S-100 world, with all of its shady vendors, has certainly proven the ability of the marketplace to sort out the quality manufacturers. Ohio Scientific has taken the contrary position of being the sole "authorized" source for bus-compatible products for its systems. This sort of position puts considerable pressure on Ohio Scientific to compete at least functionally with the rest of the microcomputer marketplace. The results of such pressure in some cases are features that compare favorably in print, but not in action. This is also why some Ohio Scientific products are not available for up to six months after they are announced in advertisements. It is very difficult for any single company to compete successfully in all facets and at all levels of the microcomputer market.

This brings us to the conclusion: Ohio Scientifc is a market-oriented company. It will do whatever is necessary to sell its product and maximize its profit. That isn't necessarily bad. Ohio Scientific has consistently exhibited a remarkable ability to respond to what the marketplace demands. Unfortunately, that same attitude also leads the company to attempt to protect its captive market with whatever means are at its disposal.

Although this may be a short-sighted attitude and second-source vendors may, in the long run, take some of the pressures off Ohio Scientific to remain competitive in all areas of microcomputing, there are no indications that Ohio Scientific is going to change its attitude towards those vendors. Partially as a result of that attitude, there is currently a lack of strong second sources for Ohio Scientific hardware. With the increased visibility and vitality of the Ohio Scientific user community, I don't think this weakness in the marketplace will continue for long.

MICRO

HI-RES FOOTBALL



The ONLY real-time action football for the APPLE computer. Play a friend or play against the computer. Either way you and your opponent call the plays and control the players movement; both passing and running. The field and all men are shown in full HI-RES graphics. Real time control is yours both on offense and defense. Fast machine language execution provides realistic action!

48K Apple or Apple Plus $39.95

See it at your dealer, or call

ON-LINE SYSTEMS

36575 Mudge Ranch Road
Coarsegold, CA 93614 • (209) 683-6858

BASIC THAT SCREEMS

Is the sedate pace of your OSI BASIC taking the fun out of your programming?

Then turn your system on to FBASIC.

Now you can compile your programs with FBASIC and take full advantage of your computers potential.

FBASIC is fast. Not just 5 or 10 times as fast as OSI BASIC. FBASIC is OVER ONE-HUNDRED TIMES FASTER! Allowing you to do unheard of things in BASIC. Things that used to require assembly-language. With no need to learn a new language.

The secret to this incredible speed is that FBASIC produces native 6502 machine code. With no run-time interpreter to get in the way of all-out machine performance.

FBASIC is good for any application from wordprocessors to video games. Whatever tickles your fancy. FBASIC accepts standard BASIC source files, and produces executable disk-based object files. It includes many of the features you have always wanted: hex constants, convenient machine language calls, optional user selection of array locations, and more.

As an example of its power and performance; FBASIC was first written in OSI BASIC, which took 6 hours to compile itself. With many features added since then, it now compiles itself in a little over 4 minutes!

So let that pent-up performance out. Find out what your machine is really capable of. Feed it some FBASIC and stand back!

FBASIC runs under OS-65D and requires 48K.

Available on 8" diskette for $150.

Please include $5 for shipping & handling.

PEGASUS SOFTWARE

P.O. Box 10014, Dept. M-2,
Honolulu, Hawaii 96816

EROM #1

Requires Applesoft Rom and Romplus.

CRAE's powerful Global Change/Find, optimized List Command, Hex to Decimal and Decimal to Hex conversion now available on a 2716 EPROM.

EROM #1 w/manual

$69.95

EROM #2

Requires Applesoft Rom and Romplus.

CRAE's Autoline numbering, formatted Memory Dump, Append, number conversion (Hex/Dec) on one 2716 EPROM.

EROM #2 w/manual

$49.95

EROM #3

CRAE's powerful renumber and quote function now on two 2716 EPROMS.

EROM #3 w/manual

$49.95

EROM 1, 2, 3...

$149.95

SEE YOUR LOCAL DEALER OR SEND CHECKS TO

HIGHLANDS COMPUTER SERVICES

14422 S.E. 132nd
Renton, Washington 98055
(206) 228-6691

Washington residents add 5.3% sales tax. Applesoft and Apple are registered trademarks of Apple Computers, Inc.

Romplus is a registered trademark of Mountain Computers, Inc.

MICRO

6502 Bibliography: Part XXVII

Dr. William R. Dial
438 Roslyn Avenue
Akron, OH 44320

744. Recreational Computing 8, No. 6, Issue 45 (May/June 1980)

Lindsay, Len, "Home Video Displays," pgs. 36-37.
General discussion of video displays.

745. The Seed 2, No. 5 (May 1980)

Taylor, Terry N. "New Programs," pg. 9-11.
The Seed librarian lists 429 new programs entered into the club library of Apple programs.

Anon, "International APPLE Core," pg. 12.
New policies of the IAC developed at the recent meeting of the IAC. Dues, access to THE SOURCE, etc.

746. SES Newsletter, Issue 18 (May 1980)

Abbott, J.H., "Animated Sphere Rotation," pgs. 7-8.
This is a very nice hi-res graphics program for the APPLE.

747. Rubber APPLE Users Group 3, No. 5 (May 1980)

Gabelman, Ken, "ONERR GOTO," pgs. 5-6.
A discussion of errors and their management, on the APPLE.

Gabelman, Ken, "Disk Structures," pgs. 7-9.
APPLE data structures, binary and Basic, are discussed.

748. APPLE Bits 2, No. 5 (May 1980)

Sanders, Dwight, "Practical Programs - and Maybe Some that Aren't," pg. 4.
Listings for APPLE Screen Editor, both Tape and Disk versions.

Martie, Ed, "Synergistic Operating Systems," pgs. 9-10.
Some notes on converting a Basic program into Pascal on the APPLE.

Goulder, Al, "Stop Watch and Lap Timer," pg. 11.
Here is a stop watch that is a little different.

Townsend, Jeff, "Helpful Hardware Hints," pgs. 13-14.
Interfacing the APPLE to read eight or sixteen switches. With hardware directions, basic and machine language listings.

Martie, Ed, "Interfacing an External Terminal to the APPLE for use with PASCAL," pg. 15.
Discussion and PASCAL software for using the Heath H-19 Terminal with the APPLE.

Anon, "IAC Application Notes," pg. 24-29.
A number of Application Notes from the International APPLE Core including: Append fix in DOS 3.2; Applesoft HIRES Routines in Applesoft ROM; Literal Input which permits you to enter commas, etc. into Applesoft print statements; Stalking the Wild Left Bracket-and reverse bracket, underline, etc.; also a list of 61 known PASCAL Problems.

749. Softside: Apple Edition 1, Issue 5 (May, 1980)

Wolfson, Mark J., "The Small Marquee," pg. 10-13.
Learn to spell without vowels, on the Apple.

Mauch, John, "Hyperboloid," pg. 16-21.
Three-dimensional HIRES graphics program.

Hartley, Chuck, "Magic Cave," pg. 18-21.
An Adventure type program to locate the legendary cave with a fortune in solid gold.

Swenson, Carl, "Right/Left," pg. 50-51.
A training program for small children to teach them the differences between "left" and "right".

Blackwood, Brian and George, "Intimate Instructions in Integer Basic," pg. 53-55.
Continuing with Lesson II of this instructional series.

750. FROM THE CORE (May, 1980)

Schmoyer, Jeff, "Now What's Wrong?," pg. 5-10.
A serial interface article with hardware notes and driver listing. Provides RS-232 EIA interface for the Apple to a Base 2 Model 800ST printer or a TI 810.

Budge, Joe, "High Speed Disk Initialization," pg. 10-13.
With the instructions in this article you can initialize Apple disks in 45 seconds instead of waiting 2 minutes.

751. Abacus II 2, Issue 4 (April, 1980)

Waxer, Daniel A., "Memory Display," pg. 7-9.
Program, in Integer Basic, for displaying Apple memory, byte by byte.

Anon., "Applesoft Hires Routines," pg. 15.
Entry points to the machine level Hires graphics.

Anon., "HIRES Screen Function Demo," pg. 16.
Graphics demo for the Apple.

752. The Seed 2, No. 3 (March, 1980)

Knaster, Scott, "Using the 'Old Monitor ROM' with the Language System," pg. 2.
Hints for using the old ROM, together with a software fix.

Foens, Bob, "Geejo," pg. 4.
Tips on easier editing, writing programs with delay loops, paddle-read program, TV interference, etc. on the Apple.

753. The Seed 2, No. 4 (April, 1980)

Foens, Bob, "Geejo," pg. 5.
How to put inverse into catalog listings, mystery program, debugging tips, for the Apple.

Mills, Craig A., "Volume Number," pg. 15.
How to change the volume number on your disk, for the Apple.

754. Apple Gram 1, No. 2 (Feb., 1979)

Carpenter, Chuck, "A Simple Disk File Program," pg. 3.
Listing and explanation of the program, for the Apple.

Matzinger, Bob, "On Printing Data in Columns," pg. 6.
Program to print information in two columns on the Apple.

755. Apple Gram 1, No. 3 (March, 1979)

Graham, Johnny, "User Key In Routine," pg. 4.
Load and Run cassettes with one key-stroke.

756. Apple Gram 1, No. 4 (April, 1979)

Ferrell, Bobbie, "Bugs and Butterflies," pg. 2.
Two graphics programs in Apple Hires.

Carpenter, Chuck, "Apple II Clock," pg. 7-8.
A machine language program for the Apple.

Sander-Cederlof, Bob, "Color Digits with Time of Day," pg. 10-11.
A clock program with graphics, Apple Lo-Res.

Santovec, Mike, "Disk JOIN Function for S-C Assembler II," pg. 13-15.
A program providing append or join functions for disk files, for the Apple.

757. Apple Gram 1, No. 5 (May, 1979)

Graham, Johnny, "Machine Code Poke Generator," pg. 3.
How to put machine language routines into your Applesoft or Integer Basic programs.

Sander-Cederlof, Bob, "Rounding in Applesoft," pg. 5.
Rounding off decimals on the Apple.

David, Jill, "Out of Sorts?; Part I," pg. 7-9.
A tutorial article on SORTS.

758. Apple-Com-Post 6, (Jan., 1980)

Schultz, H.J., "H64 Ham-Interface A650 Deluxe RTTY," pg. 4.
An interface for the Apple II permitting RTTY in ASCII or BAUDOT.

759. Applesauce 2, No. 1 (Jan., 1980)

Hyde, Randy, "The Assembly Line," pg. 14-16.
Using RWTS, an Example—Fake Catalog.

Hyde, Randy, "The Apple Monitor: Video Display Variables," pg. 25-26.
The third part of a continuing series describes zero page locations used by the Apple Monitor.

Wayne, Phil, "APscaLLE: A 'Super' PASCAL," pg. 29-30.
A special language for the Apple based on PASCAL.

760. Apple Gram 1, No. 6 (June, 1979)

Sander-Cederlof, Bob, "Copying Binary Disk Files," pg. 3.
How to find the starting address and length of binary files on 16K, 32K, and 48K Apples.

Carpenter, Chuck, "Apple-80 Simulator," pg.4-5.
A program to simulate the 8080 microprocessor on the Apple.

David, Jill P., "Out of Sorts: Part II," pg. 10-14.
The second article in a series on SORTS.

761. Apple Gram 1, No. 7 (July, 1979)

Carpenter, Chuck, "Apple II's Three M's: Part I," pg. 5-6.
Memory, Monitor and Machine Language are explained in a tutorial series.

Matzinger, Bob, "Formatting in Applesoft," pg. 14.
Lining up the decimal points in Applesoft programs.

David, Jill, "Out of Sorts: Part III," pg. 15-19.
This third article describes QUICKSORT for the Apple.

762. Sym-Physis No. 1 (Jan./Feb., 1980)

Anon., "Relocate for the SYM-1," pg. 7-12.
Discussion and listing of a relocate program.

Gettys, Thomas, "Merge/Delete Program for SYM Basic," pg. 8.
Machine language program for the SYM-1.

Du Peu, Maurie, "A Program to Display SYM-1 LED Segment Codes," pg. 18.
A machine language program for the SYM-1.

Wells, George, "Suggested Hardware Modification," pg. 18.
Add more PROMS to your SYM-1 board.

763. Apple Gram 1, No. 8 (Aug., 1979)

Sander-Cederlof, Bob, "Euclid's Algorithm," pg. 4.
Short explanation and demo listing illustrating "algorithm"

Laumer, Mike, "How to Get That Weird and Crazy Program Saved to Run from Disk," pg. 6-8.
How to save 'difficult' tape programs to disk on the Apple.

Carpenter, Chuck, "Apple II's Three M's: Part II," pg. 10-12.
This month's installment discusses the Apple Monitor.

764. Apple-Com-Post No. 7 (Feb., 1980)

Goetzke, Uwe, "Pascal-eine Ein-furung," pg. 16-18.
Pascal Discussion and program for Apple.

Barbieri, Nino, "Program Kniepe," pg. 20.
Short demo graphics program in Apple Hi-Res.

765. Apple Gram 1, No. 9 (Sept., 1979)

Carpenter, Chuck, "Apple II's Three M's: Part III," pg. 4-7.
The third M, Machine Language, is discussed in this tutorial on the Apple.

Broderick, John, "Strings in Integer Basic," pg. 15.
A short tutorial on strings for the Apple Integer Basic.

766. Apple Gram 1, No. 10 (Oct., 1979)

Marsh, Bob, "Simple Apple Serial Interface," pg. 9-11.
Hardware diagram and software listing to interface the Heathkit H-14 printer to the Apple II.

767. Apple Gram 1, No. 11 (Nov., 1979)

McClelland, George, "Simple Pascal Text Formatter," pg. 7-8.
Create a text file in the editor, then print it out on the printer with upper and lower case using this Apple Pascal program.

Sander-Cederlof, Bob, "Apple DOS Version 3.2.1," pg. 11.
Some comments on the new version of Apple DOS.

Carpenter, Chuck, "Apple II's Three M's: Part IV," pg. 12-14.
This time the author of this series discusses another Apple feature, the Mini-Assembler.

768. Apple Gram 1, No. 12 (Dec., 1979)

Sander-Cederlof, Bob, "Tiny Program Displays Text Files," pg. 2.
Program makes it easy to read those Apple Text Files.

Matzinger, Bob, "SSM AIO Card," pg. 8.
Details on how to use the new AIO card from Solid State Music (SSM) to interface the Apple with the Heathkit H-14 Printer.

769. Apple Gram 2, No. 1 (Jan., 1980)

Walston, Joe, "Calendar Generator," pg. 4-5.
An Apple program for day of week, calendar, etc.

David, Jill, "Programmer's Tools," pg. 7-9.
A tutorial on the push-down stack together with a demonstration listing.

Matzinger, Bob, "Fancy Listings," pg. 10-13.
How to list your Apple Basic programs on a full width of paper and print a complete word without splitting it.

770. Apple Gram 2, No. 2 (Feb., 1980)

Broderick, John, "Please Pass the Pascal," pg. 11.
Notes on getting started in Pascal.

Zant, Robert F., "ON ERR -- Punt?", pg. 12.
Routine for intercepting error conditions using the Apple monitor.

Sander-Cederlof, Bob, "Saying HELLO with Style," pg. 14-15.
A snazzy "Hello" program for the Apple Disk.

771. Sym-Physis, No. 2 (Mar./Apr., 1980)

Brown, J.W., "Ultra Renumber for BAS-1," pg. 4-8.
An automatic renumbering program for the SYM-1.

Rinard, Phillip M., "Doodling with the KTM-2," pg. 9-15.
The KTM-2 keyboard Terminal module provides a convenient way to add a video interface to a SYM-1 as well as a keyboard.

Bach, Stephen E., "Clock Program for SYM-1 with Terminal," pg. 17-19.
A clock program converted to RAE format for the SYM-1.

Luxenberg, H.R., "24-Clock for SYM-1," pg. 20-22.
A simplified version of the previous clock version.

772. Apple Gram 2, No. 3 (March, 1980)

Sander-Cederlof, Bob, "Fancy Hello Program for Applesoft," pg. 4.
A still fancier "Hello" program for the Apple.

Firth, Mike, "More Tokens and Errors," pg. 5.
Here is a program to get you started examining BASIC programs to find memory locations that are giving you errors.

Bowser, Bob, "PRINTIT, a Pascal Program for Lower Case Output," pg. 11-14.
Take advantage of the Apple keyboard and the Pascal editor to do limited text processing with this listing.

Sander-Cederlof, Bob, "Applesoft precision," pg. 16-17.
What goes on in Applesoft and what to do if you need greater precision, up to 21 digits.

Firth, Mike, "Banner Routine," pg. 17.
A subroutine which permits you to display a lot of data on a single line of the screen.

773. Apple-Com-Post, No. 8 (April, 1980)

Schultz, Heinz Juergen, "Microphone Amplifier for the Apple II Cassette Input," pg. 9.
Two transistor amplifier makes possible use of a dynamic microphone.

Anon., "Software Tips," pg. 10-13.
A fix for a DOS error; saving High-Resolution pictures to cassette; packing Applesoft programs.

774. Apple Gram 2, No. 4 (April, 1980)

Firth, Mike, "Finding Tokens in Your Programs," pg. 6.
Examine your own programs in memory.

Hatcher, Rich, "A Joystick and an Extra Switch Input," pg. 10-13.
Hardware addition to the game output port on the Apple.

Sander-Cederlof, Bob, "Calculation of PI," pg. 15-16.
Several programs, fast and slow, for calculating PI.

775. Applesass 2, No. 4 (May, 1980)

Fred, Dennis, "Hidden Character," pg. 4.
This short program will display all control characters as inverse flashing in your Apple Catalog.

Anon., "Intermediate Applesoft," pg. 5.
Handling error messages on the Apple.

Edelstein, Arnold, "USR Function Demo," pg. 6.
Assembly language program for the Apple.

Anon., "Advanced Applesoft," pg. 7.
Apple commands USR, Call and Ampersand (&).

Buchler, Dan, "Justification Routine," pg. 9.
This routine will right justify all output to the screen.

Edelstein, Arnold, "Floating Point Tutorial," pg. 10.
This program takes decimal data entered at the Apple keyboard and returns its floating point equivalent as well as the binary equivalent of the first two bytes of the mantissa.

Edelstein, Arnold, "Wait a Bit," pg. 11-12.
Explanation of how floating point numbers are configured.

776. Robert Purser's Magazine (Spring, 1980)

Purser, Robert, "Purser's Magazine."
This magazine is the successor to "Computer Cassettes Review." It covers software for the Apple, Pet and Atari as well as Level II TRS-80. Disk programs are now included. Detailed discussions of programs are given.

777. Peek(65) 1, No. 4 (May, 1980)

Bonser, R.E., "Graphics Routine," pg. 7.
Routine for OSI computers useful in game programs.

Peabody, Al, "Framing Errors," pg. 10.
How to avoid Error 17---Framing Error, on your OS 65 U.

Lucas, J.E., "String Handling Problem," pg. 11.
String Handling problem in Microsoft BASIC for OSI computers.

Sanders, Jim, "The Exterminator," pg. 13.
A bug-correction program for the OS-65U Disk.

Sanders, Jim, "The 8K BASIC 'Wait' Instruction," pg. 14.
Discussion of the WAIT instruction and an example listing for OSI computers.

778. The Cider Press (May, 1980)

Thompson, C.J., "The Executive Branch," pg. 6-7.
A tutorial on this useful function with 3 examples.

Fields, Randy, "Screen Disk Commands," pg. 8.
Hints on using the Apple disk program.

Post, Steve, "Edit + Corrections," pg. 8-9.
Fixes for problems for the EDIT+ program, for the Apple.

Sokal, Dan, "Pascal-PEEKs and POKEs," pg. 10.
Program designed to be added to the Pascal System Library. Includes sample subroutine for keypress.

Tyro, A., "Linelimit," pg. 11.
A Pascal program for the Apple producing a screen pause when 20 lines of text appear on the screen.

779. The Grape Vine (May, 1980)

Lawson, Stephen M., "Sort Routines," pg. 2-3.
A Foote sort routine stripped of the custom features so it can have general use. Also given is a Singleton sort. A demo program for illustrating each is given.

Lawson, Stephen M., "Lores POKE Demo," pg. 4.
Illustrates another method of getting colors to the Lo-Res screen on the Apple.

Lawson, Stephen M., "Easter Calendar," pg. 5-7.
Updated calendar program including modifications for Ascension Day, Pentecost Sunday, etc.

780. Abacus II 2, Issue 5 (May, 1980)

Robbins, Greg, "Telephone Dialer," pg. 6-7.
A dialer system that does not require a modem, only the listed software and a simple relay for the phone line.

Freeman, Larry L. and Davis, James P., "Applesoft Menu," pg. 8.
Automatic menu for running Catalog programs on the Apple Disk.

Robbins, Greg, "Hires Eraser," pg. 9.
Use this erase routine at the end of a program using the Apple Hi-Res screen.

Apple Staff, "Apple Fix in DOS 3.2 & 3.2.1," pg.9.
The fix involves writing a missing 'end of file' marker.

IAC Staff, "Application Note: List of Known Pascal Problems," pg. 11-14.
IAC will release Application Notes from time to time. Updates and Fixes for this list of Pascal problems will be released later.

IAC Staff, "Application Note: Literal Input," pg. 15-16.
A routine for Applesoft which allows you to enter commas, quotes and colons without getting an 'extra ignored' error.

IAC Staff, "Application Note: Stalking the Wild Left Bracket," pg. 17-18.
Apple keyboard changes to permit typing left bracket, reverse slant and underline.

Robbins, Greg, "Modifying Your Apple to Accept User Firmware," pg. 19.
Hardware Mod to the Apple board so that it will accept your own EPROMS.

Davis, James P., "Lister Writer," pg. 20.
Basic and machine language programs to enable listing on a Trencom 200 printer.

781. Stems From Apple 3, Issue 5 (May, 1980)

Hertzfeld, Andy, "Andy's Switch," pg. 5-6.
Two programs that allow you to have two completely different Catalogs in a single Apple diskette.

Capella, Mark, "Hide," pg. 7.
This program for the Apple hides program names from appearing on the disk catalog.

Noble, Geoffrey E., "Hi-Res Screen Dump," pg. 12-13.
This is an updated version of the Integral Data Hi-Res Screen Dump for the IDS 440 Printer.

Stein, Dick, "File Cabinet Changes," pg. 16.
Some corrections to the previously updated version published in the January issue of Stems From Apple.

782. Apple Gram 2, Issue 5 (May, 1980)

Broderick, John, "Pull From the Stack to Get Back," pg. 8.
Program for the Apple to show how the stack saves the return address for JSR to a subroutine.

Matzinger, Bob, "Auto Dial," pg. 10-12.
A modified program in which data statements for new listings are added automatically to the program.

Sander-Cederlof, Bob, "Some Corrections to the Orchard," pg. 13.
Corrections for the important article "Applesoft Internal Entry Points" published in the initial issue of the IAC Apple Orchard.

Hatcher, Rich, "What? Another Stop-List Program?", pg. 14-15.
This one uses CTRL-S on the Apple to change speeds.

783. Apple Cookbook 1, No. 3 (May/ June, 1980)

Jones, Tad, "Tony's Read Program," pg. 8-9.
A utility program for disk catalogs, etc.

784. Call-Apple 3, No. 4 (May, 1980)

Hertzfeld, Andy, "Andy's Switch," pg. 7-17.
Put two different catalogs on a single disk side.

Lingwood, Dave, "The Return of the Mysterious Mr. Ampersand," pg. 26.
You can use the Ampersand in many interesting ways.

Gilder, Jules H., "Printer Fix for Parallel Interface with a Centronics Printer," pg. 36.
This fix is for a problem in which the Centronics Printer ignores a carriage return at the beginning of a line on the Apple.

Kipperman, Steven M. et. al., "File Cabinet Update," pg. 41-43.
Several comments, modified listings, etc. for File Cabinet.

Capes, Nelson R., "File Cabinet," pg. 44.
Adding input data to this oft modified and extended program.

Huelsdonk, Bob, "Making Basic Behave: Part II," pg. 48-49.
A utility for programmers, Part II.

Adams, Steve, "Single Drive Copy," pg. 51-52.
Copy your Apple disks with only one disk drive using this useful utility.

Wagner, Roger, "Booting Binary Programs," pg. 53.
How to use a binary program as the 'Hello' Program in booting a diskette on the Apple.

785. Apple-Com-Post, No. 9 (May, 1980)

Knuelle, Alfred "Hardware Tips: Paddles, Joysticks, Etc.," pg. 8.
All about accessories for the Game Port output on the Apple.

786. Sym-Physis, Issue 3 (May/June, 1980)

Sinnet, Jay C., "Hardware Modification for Better Tape Reliability," pg. 4.
A hardware modification for the SYM-1 board to improve cassette input reliability.

Luxenberg, H.R., "Cassette Recorder Tips," pg. 3.
Miscellaneous tips on increasing reliability of the SYM-1 cassette subsystem.

Cyr, Jean M., "A Sorting Patch for RAE," pg. 5-6.
A portion of "User Patch for RAE-1" which permits the printing of an alphabetically sorted Label File.

Anon., "Graphics Demonstration Package," pg. 7-9.
A graphics program in Basic, for the SYM-1.

Gowans, Bill, "Hi-Density Plotting with the KTM-2," pg. 11-18.
This routine effectively quadruples the KTM-2 graphics density by mapping a virtual 48x160 screen onto the real 24x80 screen.

Luxenberg, H.R., "Scope Graphics and Computer 'Generated' Music," pg. 12-21.
A couple of novelty demo programs and some music utility programs. Also some discussion of using Hal Chamberlin's approach to music generation on the SYM-1.

Thompson, Bruce, "Basic and the 2K Symbolic Assembler," pg. 23.
A short program called by BASIC's USR to dump or load specific memory locations.

Anon., "Inexpensive D/A Converter," pg. 24.
Hardware project based on a 4050 chip.

787. The Target (May/June, 1980)

Bresson, Steve, "Renumber," pg. 2-3.
A basic program for the AIM 65 to renumber BASIC programs.

Lowery, Dale, "Subroutines," pg. 4.
Some assembly routines used to get individual or specific characters in the AIM 65 display.

Roberts, Steve, "RS-232," pg. 6.
An interface for running a DECwriter at 300 baud with an AIM.

788. Peelings II 1, No. 1 (May/June, 1980)

Staff, "Peelings II, pg. 2.
Peelings II is a new magazine devoted to the evaluation of Apple II software. The first issue reviews in some detail eight game programs, seven technical programs and several major utilities, etc.

789. Apple Shoppe 1, No. 6 (May/June, 1980)

Anon., "Circuit Theory," pg. 7-8.
An electrical program in Pascal for the Apple.

Anon., "Graphics in Assembly, Part II," pg. 8.
A tutorial on Graphics machine language.

Anon., "Assembly, Part II," pg. 9-16.
A tutorial leading into a development example of a Text Editor.

Crouch, Bill, "Down to Business," pg. 18-20.
Handling of Disk Files with a demo routine.

Amromin, Joel L., "Pascal Linefeed Mod for AIO, etc.," pg. 23-24.
A mod for the Solid State Music AIO peripheral card for the Apple.

Russell, P. Tim, "Pascal Workshop," pg. 24-25.
Rounding Numbers in Pascal.

790. Nibble, No. 3 (May/June, 1980)

Harvey, Mike, "Text Processing with TOUGH," pg. 7.
Text outputter, updater, and generalized handler for word processing on the Apple II.

Harvey, Mike, "Apple II 'Paper Tiger' Graphics,", pg. 15-17.
Subroutines to take the page 1 graphics of the Apple and convert it for printing on the Paper Tiger.

Mottola, R.M., "Machine Language Screen Dump," pg. 19.
This screen dump routine will allow you to print all or part of the Apple screen.

Floeter, Alan D., "Applesoft REM Remover- Or, Now You See It, Now You Don't," pg. 20.
Decrease the memory space required and the execution speed by removing those REM statements from your Applesoft listing.

Connolly, Rick, "Improving the Multiple Array Sort," pg. 26-27.
How to realize the speed advantage in sorts such as the Shell-Metzner sort.

791. Rubber Apple User Group Newsletter 3, No. 6

Gabelman, Ken, "Disk Structures II," pg. 4.
Tutorial on random files as applied to a mailing list.

792. The Harvest 1, No. E (July, 1980)

Allen, Earl, "Applesoft Fast File Accessing," pg. 1-3.
A fast method of handling massive amounts of data is described for the Apple disk system.

Stadfeld, Paul, "Turtleshapes," pg. 7-8.
A novel graphics program in Apple Hi-Res using some of the principles of Pascal Turtlegraphics.

793. The River City Apple Corps Newsletter (June, 1980)

Wyman, Curt, "Hi-Res Graphics: Rotating Three Dimensional Shapes," pg. 2-4.
A graphics tutorial.

Sethre, Tom, "Roots," pg. 4-6.
A tutorial on machine language loops using the 6502.

794. The Seed 2, No. 6 (June, 1980)

Rahn, Ray, "Ray's Apple Crib Sheet," pg. 10-13.
Basics in the operation of the Apple Disk system.

795. Cider Press (June, 1980)

Anon., "Apple Tricks," pg. 6.
A patch to DOS 3.2 of the Apple disk system which will make it INIT a disk in half the time and boot up in 2 seconds.

796. Abacus II 2, Issue 6 (June, 1980)

Wine, Hal, "Apple II ROM Monitor Subroutine List," pg. 4-8.
Discussion of a variety of monitor subroutines.

Freeman, Larry, "The Hole in Apple's Integer Numbers," pg. 8-10.
A bug, a fix and an instructive explanation of it all.

Wine, Hal, "Color Generation in Hi-Res," pg. 12-14.
An explanation of how the Apple Hi-Res colors are generated.

Letterbox
(continued from page 59)

Dear Editor:

I am an AIM 65 user and I have written a few assembly language programs that may be of interest to others. I would like to sell these programs, but I don't know much about this.

Would it be better to just write an ad or write an article, say for MICRO, describing these programs with an offer to sell software on cassette, or what? How does one establish a market value? Should just the object code or both source and object be released?

Alan M. Davis
RFD #2130, Rt. 106
Syosset, New York 11791

You will find a detailed discussion of software distribution on the editorial page of our October and November issues (29:5 and 30:5).

If an article on software is to be considered for publication in MICRO, the article must include the software printout, not merely a description of it. In this case, the author offers his software to the public without charging for it.

If an author wishes to sell software, through the medium of MICRO, he has three ways of doing so. He can queue up for a free listing in The MICRO Software Catalog, which appears in each issue; he can submit a classified ad; or he can place a display ad. We have a backlog of listings for The MICRO Software Catalog, so an author will have to wait his turn. (His listing must also follow our strict format.) A classified ad will be published immediately, but is limited in size and attracts a mini-mum amount of attention. Display ads cost more, but offer the greatest freedom to advertisers—a variety of possible sizes, flexibility in content, design, typography, and placement within the publication. They can also be scheduled to coincide with the dates when software is ready for release. In addition, reduced rates are available for those who advertise frequently.

On the subject of usefulness—both the source and object codes are more useful to purchasers than just the object code.

The Editor

MICRO

Statement of Ownership, Management and Circulation

Required by the act of Congress of October 23, 1962, of MICRO, published monthly at Chelmsford, Massachusetts for September 1980:

The name and address of the publisher is **MICRO INK, Inc.**, 34 Chelmsford Street, Chelmsford, Massachusetts. The Editor/Publisher is Robert M. Tripp of Chelmsford, Massachusetts.

The owner is **MICRO INK, Inc.**, Chelmsford, Massachusetts, and the names and addresses of stockholders owning or holding one percent or more of the total amount of stock are: Robert M. Tripp and Donna M. Tripp of Chelmsford, Massachusetts.

The known bondholders, mortgagees and other security holders owning one percent or more of the total amount of bonds, mortgages or other securities are: none.

The average number of copies of each issue of this publication sold or distributed through the mails or otherwise to paid subscribers during the twelve months preceding the date shown above is: 10,521.

I certify that the statements made by me above are correct and complete.

Signed: Robert M. Tripp
Editor/Publisher

ADVERTISERS' INDEX

December 1980

The home computer you thought was years away is here.

C8P DF

Ohio Scientific's top of the line personal computer, the C8P DF. This system incorporates the most advanced technology now available in standard configurations and add-on options. The C8P DF has full capabilities as a personal computer, a small business computer, a home monitoring security system and an advanced process controller.

Personal Computer Features
The C8P DF features ultra-fast program execution. The standard model is twice as fast as other personal computers such as the Apple II and PET. The computer system is available with a GT option which nearly doubles the speed again, making it comparable to high end mini-computer systems. High speed execution makes elaborate video animation possible as well as other I/O functions which until now, have not been possible. The C8P DF features Ohio Scientific's 32 x 64 character display with graphics and gaming elements for an effective resolution of 256 x 512 points and up to 16 colors. Other features for personal use include a programmable tone generator from 200 to 20KHz and an 8 bit companding digital to analog converter for music and voice output, 2-8 axis joystick interfaces, and 2-10 key pad interfaces. Hundreds of personal applications, games and educational software packages are currently available for use with the C8P DF.

Business Applications
The C8P DF utilizes full size 8" floppy disks and is compatible with Ohio Scientific's advanced small business operating system, OS-65U and two types of information management systems, OS-MDMS and OS-DMS. The computer system comes standard with a high-speed printer interface and a modem interface. It features a full 53-key ASCII keyboard as well as 2048 character display with upper and lower case for business and word processing applications.

Home Control
The C8P DF has the most advanced home monitoring and control capabilities ever offered in a computer system. It incorporates a real time clock and a unique FOREGROUND/BACKGROUND operating system which allows the computer to function with normal BASIC programs at the same time it is monitoring external devices. The C8P DF comes standard with an AC remote control interface which allows it to control a wide range of AC appliances and lights remotely without wiring and an interface for home security systems which monitors fire, intrusion, car theft, water levels and freezer temperature, all without messy wiring. In addition, the C8P DF can accept Ohio Scientific's Votrax voice I/O board and/or Ohio Scientific's new universal telephone interface (UTI). The telephone interface connects the computer to any touch-tone or rotary dial telephone line. The computer system is able to answer calls, initiate calls and communicate via touch-tone signals, voice output or 300 baud modem signals. It can accept and decode touch-tone signals, 300 baud modem signals and record incoming voice messages. These features collectively give the C8P DF capabilities to monitor and control home functions with almost human-like capabilities.

Process Controller
The C8P DF incorporates a real time clock, FOREGROUND/BACKGROUND operation and 16 parallel I/O lines. Additionally a universal accessory BUS connector is accessible at the back of the computer to plug in additional 48 lines of parallel I/O and/or a complete analog signal I/O board with A/D and D/A and multiplexers.

Clearly, the C8P DF beats all existing small computers in conventional specifications plus it has capabilities far beyond any other computer system on the market today.

C8P DF is an 8-slot mainframe class computer with 32K static RAM, dual 8" floppies, and several open slots for expansion.

Prices start at under $3,000.

Computers come with keyboards and floppies where specified. Other equipment shown is optional.

For literature and the name of your local dealer, CALL 1-800-321-6850 TOLL FREE.

OHIO SCIENTIFIC
1333 SOUTH CHILLICOTHE ROAD
AURORA, OH 44202 • (216) 831-5600